April 2016

Simulation of Diffusion in Cellular Geometries Using GPUs

Cem Mehmet Unsal
Worcester Polytechnic Institute

Project Number: MQP ERT - 15Q1

DATA PARALLEL SIMULATION OF FLUORESCENT MOLECULE DIFFUSION IN ARBITRARY CELL GEOMETRIES USING GPUS

Major Qualifying Project Report completed in partial fulfillment of the Bachelor of Science degree at

Worcester Polytechnic Institute

Cem M. Unsal

April 26, 2016

Professor Erkan Tüzel, Advisor

Department of Physics

Department of Computer Science

Abstract

Fluorescence Recovery After Photobleaching (FRAP) is a commonly used technique in cell biology for the analysis of diffusion in cellular systems. In this approach, a molecule of interest is labeled with a fluorescent protein, e.g. Green Fluorescent Protein (GFP), and is photo-bleached using a laser in a chosen area. The diffusion of labeled but unbleached molecules to this area produces a recovery of fluorescence which can then be used to calculate a diffusion coefficient. This calculation, however, requires knowledge of the underlying behavior of the molecules in the given geometry. To be able to calculate diffusion coefficients in complex cellular geometries where there is no existing analytic description, computer simulations of diffusion and realistic modeling of the imaging system is necessary. In this project, we developed a 3D computational Brownian dynamics model of the diffusion and FRAP process in arbitrary cellular boundaries, using GPUs. The model organism used is the moss *physcomitrella patens*, and the optics of image generation and laser photo-bleaching of a confocal microscope is included to produce realistic image stacks to be used in analysis. The developed computational model is cross-platform and architecture independent, and uses data parallel Java with Aparapi OpenCL bindings, producing significantly faster runtimes compared to CPU. Finally, using this approach, we investigated the effects of cell shape, bleach location, and size during FRAP process, and our results show that the apparent diffusion coefficients are strongly influenced due to these geometric effects.

Acknowledgements

I would like to acknowledge James Kingsley for his help in this project, other Tüzel Group members for helpful comments and discussion, Professor Erkan Tüzel for advising and Professor Luis Vidali and the members of the Vidali Lab for providing data and guidance.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 MOTIVATION

It is important to understand diffusion in cells in order to develop a better understanding of intra-cellular transport and cell growth. For instance, plant cells grow by carrying building materials to the tip of the cell to support growth (as illustrated in Figure 1). World crop demand is expected to grow by 48% by 2050 [1]. Building a mechanistic understanding of plant cell growth is, therefore, very important to keep up with the need to feed a growing population.

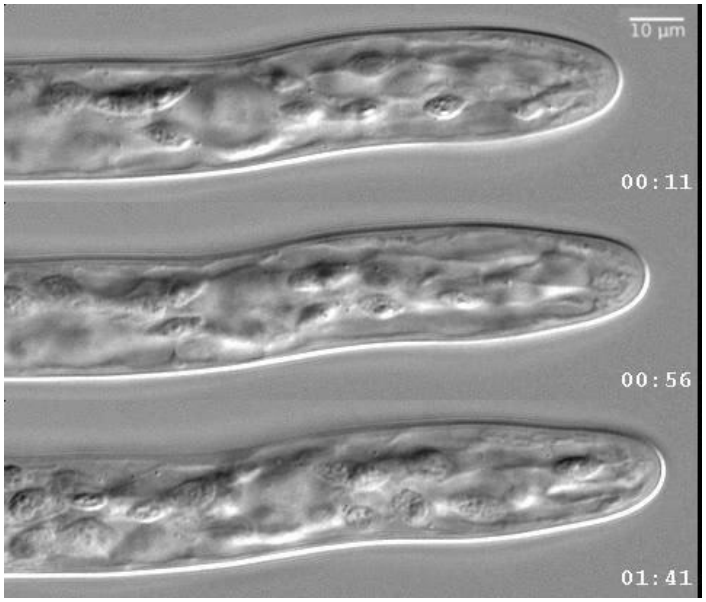

Figure 1. *Time lapse image of the growth of a moss cell. (Courtesy: Vidali Lab, WPI)*

There are several fluorescence microscopy based techniques that can be used to measure diffusive properties of molecules inside the cell. Typically, in all these approaches, the molecule of interest is labeled with a fluorescent marker. Such a fluorescent marker acts like a light source when low intensity light is shined on it.

Single particle tracking is one of these methods. In this approach, the trajectory of a fluorescently labeled molecule is tracked and mapped in order to estimate the diffusion coefficient. Multiple particle trajectories are recorded in order to calculate an average mean square displacement (MSD) and obtain a value of the diffusion coefficient [2].

Fluorescence Correlation Spectroscopy (FCS) is another method that can be used to measure the diffusion coefficient. It compares the florescent intensity between different points in time in order to calculate the diffusion coefficient [2].

Fluorescence Recovery after Photobleaching (FRAP) is also a technique that can be used to observe particle diffusion inside the cell. Shining a low intensity laser on a cell that has fluorescent molecules in causes them to fluoresce, showing the density distribution of particles inside that have this property. Shining a high intensity laser instead causes these molecules to lose their fluorescent

properties due to chemical transformations, a process called *photobleaching*. This property of fluorescent molecules can be utilized to create dark spots in the cell. The cell has a fluid medium which means that the particles are in motion, and as a result, the particles with an active florescent marker in other parts of the cell diffuse into these dark regions. During this equilibration process, the bleached molecules also move to other parts of the cell. A recovery in the intensity of fluorescent molecules in the photobleached region will be observed as shown in Figure 5.

1.2 FRAP Technique

The primary technique we used in this project is FRAP. A typical FRAP procedure consists of the following steps:

i. Image the cell to get a baseline of fluorescence intensity (I) using a low intensity laser
ii. Bleach a chosen area with a high intensity laser (this area is called a Region of Interest, ROI)
iii. Image the cell at regular time intervals

In this project, we used a confocal scanning microscopy setup. Confocal microscopy is a noise reduction technique that is used to better focus on a given focal plane. In conventional digital cameras, all of the pixels capture the image at the same time. In scanning microscopy imaging is done one pixel at a time. For each pixel, the confocal microscope pulses a low intensity laser above the area the pixel is intended to capture and records the intensity data, and form an image. This is the imaging step, and is illustrated in Figure 2.

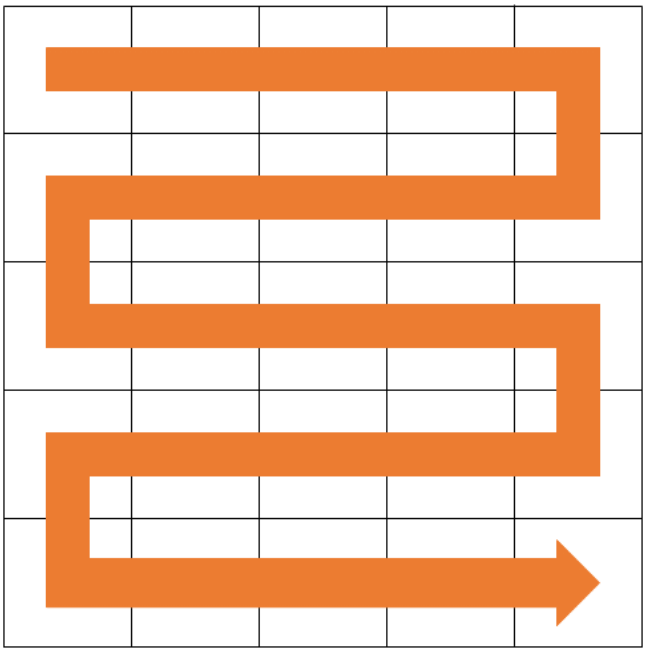

***Figure 2**. Example scan of the microscope over 25 pixels.*

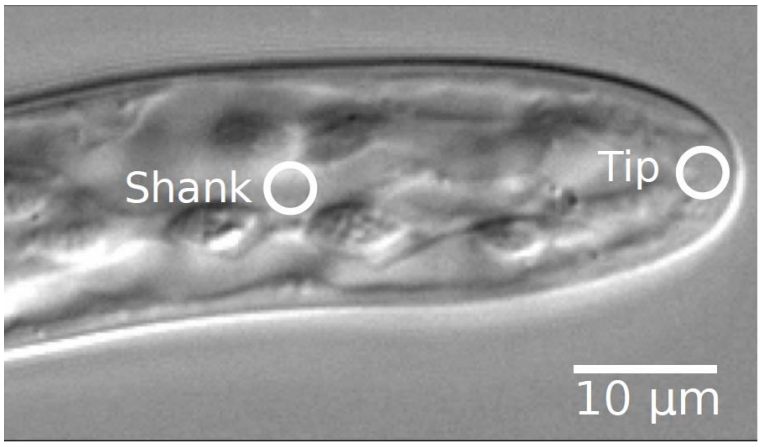

***Figure 3**. Two example ROI choices in the sample moss geometry. (Courtesy: Vidali Lab, WPI)*

In the bleaching step, a Region of Interest (ROI) is bleached, a two dimensional disk as shown in Figure 3. Sometimes the ROI is larger than the width of the light beam. In order to cover the entire ROI, multiple adjacent laser pulses are used. The type of FRAP experiment that this project is simulating uses a circular ROI, which is approximated by multiple high intensity pulses, in a similar scanning fashion as shown in Figure 4.

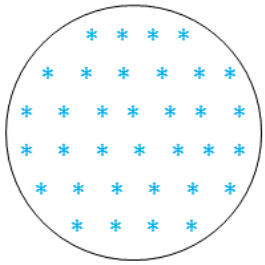

***Figure 4**. Illustration of multi-pulse bleaching.*

The recovery curve, $f(t)$, is defined as the time function of the integration of intensity over the ROI (radiant flux as a function of time), i.e.,

$$f(t) = \iint_{ROI} I\,(x, y, z, t)\,dxdy \quad , \tag{1}$$

where I is the intensity, and z is the location of the focal plane. As discussed before, confocal scanning microscope provides images as output. These output images are analyzed to construct recovery curves. Since image file has discrete data points rather than a continuous spectrum, the brightness of every pixel inside the ROI is summed. This sum is plotted as a function of time. Since baseline brightness might be different for each experiment, a few image frames before the bleaching process are used to establish this baseline. Later, all of the values can be divided by this baseline to normalize the recovery curve to unity. In the rest of this project, all of the recovery curves are assumed to be normalized in this way unless noted otherwise. For further details about the experimental system and microscopy setup, the reader is referred to Ref. [3]. Example

recoveries of ROIs at the apex and shank of a moss cell are shown in Figure 5 for illustration of the approach (taken from Ref. [3]).

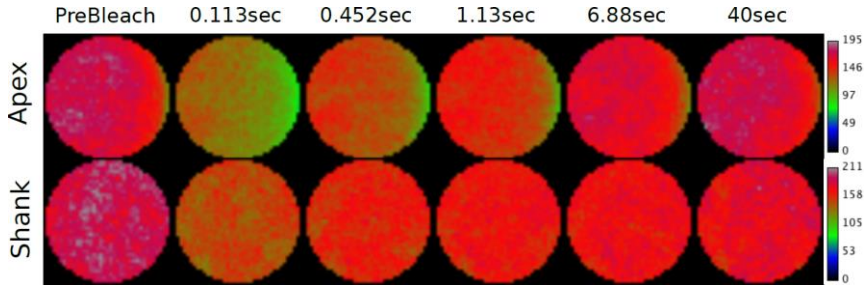

***Figure 5**. Experimental ROI recovery at the cell tip and shank. (not normalized) [3].*

1.3 Analytical Approaches

The motile behavior of fluorescent molecule density inside a domain can be described by the continuum diffusion equation, namely,

$$\frac{\partial \rho}{\partial t} = D\nabla^2 \rho, \qquad (2)$$

where $\rho = \rho(\vec{r}, t)$ is density of fluorescent molecules, D is the diffusion coefficient that is assumed to be constant, t is the time, and $\vec{r}$ is the Cartesian position vector $\vec{r} = \vec{r}(x, y, z)$.

For a few simple geometries, an analytical solution exists for the recovery curve. For example, in the case of two dimensions, in an open boundary where a circular ROI is bleached uniformly and the intensity outside ROI is assumed to be uniform and higher than the intensity within the ROI, the analytical solution for the recovery curve is

$$f(t) = 1 - \phi + (\phi e^{-\alpha})\big(\mathbb{I}_0(\alpha) + \mathbb{I}_1(\alpha)\big) \quad . \qquad (3)$$

Here $f(t)$ is the recovery curve, ϕ is the intensity inside the ROI divided by intensity outside, $\mathbb{I}_0$ and $\mathbb{I}_1$ are the modified Bessel functions of the first kind, and $\alpha = w^2/(2Dt)$, where w is the radius of the ROI.

In reality, cellular geometries are complex and the open boundary solution often gives a very different recovery curve compared to what is generated by the FRAP process. The shape and the choice of ROI, the finite scan rate of the microscope and other properties of the microscope such as non-zero width of Point Spread Function (PSF) can affect the recovery curve as well here. No analytical solution exist for many of the realistic cellular geometries that are often encountered in biology.

For this project, using our group's work on computational modeling of FRAP in simple moss geometries [3] as a starting point, we developed a computational package that simulates realistic geometries and able to handle arbitrary choices of experimental setup including

- Complex cell geometries
- Microscope properties
- Imaging sequences

- ROI properties

while also being cross-platform (hardware and operating system) and fast in order to compare to experimental data for a molecule with unknown diffusion coefficient. Our computational tool also uses GPUs to accelerate computational speed.

1.4 Summary

In this chapter we introduced FRAP and different fluorescent imaging techniques. The organization of the remainder of this MQP report is as follows. Chapter 2 describes the physical models used and the implementation details. Chapter 3 covers testing and benchmarking. Chapter 4 shows the effect of boundary and ROI choices on the recovery curve. We end this document with a summary and proposed future directions.

2 THEORY AND IMPLEMENTATION

2.1 BROWNIAN MOTION

The diffusive behavior of molecules can be modeled in two ways: by solving the diffusion equation explicitly, or by using a stochastic Brownian motion model that utilizes a large number of particles. Brownian motion refers to the motion of a particle inside a fluid medium due to elastic collisions with other particles. We used the second approach here as the model in this project because it makes it possible to more effectively simulate optical effects.

In our model, Stokes limit is assumed therefore the viscous forces are dominant over inertial forces. In the Stokes limit, the sum of the forces on each particle for each direction is approximately zero, i.e.

$$\sum \vec{F} \approx 0 \quad . \tag{4}$$

The possible forces that might be present are the external force, $\overrightarrow{F^E}$ (gravity, light pressure etc.), Brownian thermal noise , $\overrightarrow{F^B}$, and the friction, $\overrightarrow{F^f}$, which yields

$$\overrightarrow{F^E} + \overrightarrow{F^B} + \overrightarrow{F^f} \approx \vec{0} \tag{5}$$

If we assume the effect of external force is negligible, we get

$$\overrightarrow{F^B} - \zeta \vec{v} = \vec{0} \quad . \tag{6}$$

Writing this as a differential equation for position, we get

$$\frac{d\vec{r}(t)}{dt} = \frac{\overrightarrow{F^B}(t)}{\zeta} \quad , \tag{7}$$

where $\vec{r}$(t) is the position of the particle, t is time, and ζ is the friction coefficient which is $\zeta = 6\pi R\eta$ for a spherical particle [4].Here R is the molecular radius and η is the viscosity. The random force $F^B(t)$ has the following properties consistent with the fluctuation-dissipation theorem [5]

$$\langle \mathrm{F}_\alpha^\mathrm{B}(\mathrm{t}) \rangle = 0 \quad , \tag{8}$$

$$\left\langle F_\alpha^B(t) F_\beta^B(t') \right\rangle = 2k_B T\, \zeta \delta_{\alpha\beta} \delta(t - t') \quad , \tag{9}$$

where α and β are components. Here k_B is the Boltzmann constant and T is temperature. Free particles obeying such model should have a mean square displacement (MSD) of $\langle x_i^2 \rangle = 2Dt$ for each component (x-shown here). Applying Euler integration to Eq. (7) gives

$$\overrightarrow{\Delta r}(t) = \frac{\overrightarrow{F^B}(t)\Delta t}{\zeta}. \tag{10}$$

Each component of $\overrightarrow{\Delta r}$ becomes a random number with zero mean and variance $2k_BT/\zeta$ where $\zeta = k_BT/D$. This can be written as

$$x_i(t + \Delta t) = x_i(t) + N(0,1)\sqrt{2D\Delta t} \tag{11}$$

for the x-component (similar equations hold for y and z components) of particle i. Here $N(0,1)$ is a Gaussian random number with zero mean and standard deviation of 1. $i = 0,1,2,3, \dots, n-1$ is the particle index and n is the total number of particles. In the remainder of this report, the term $N(0,1)\sqrt{2D\Delta t}$ will be called a "Brownian Kick".

2.2 Collisions with the Boundaries

In the model, the cell shape is described by a triangulated surface (see Section 2.4.2). We use a set of triangular mesh elements to approximate the cell surface. When a particle receives a Brownian kick extending out of this cell boundary provided by the mesh, instead of traveling outside the cell the particle collides with the cell, boundary and bounces back. The point of intersection can be obtained by calculating the intersection between the triangle mesh element and line segment path of travel. A triangle in an arbitrary position in space can be described with three vectors. The first one describes the position of the first vertex (the choice of first vertex is arbitrary and does not affect the results) and the other two describe the positions of the other vertices with respect to the first one.

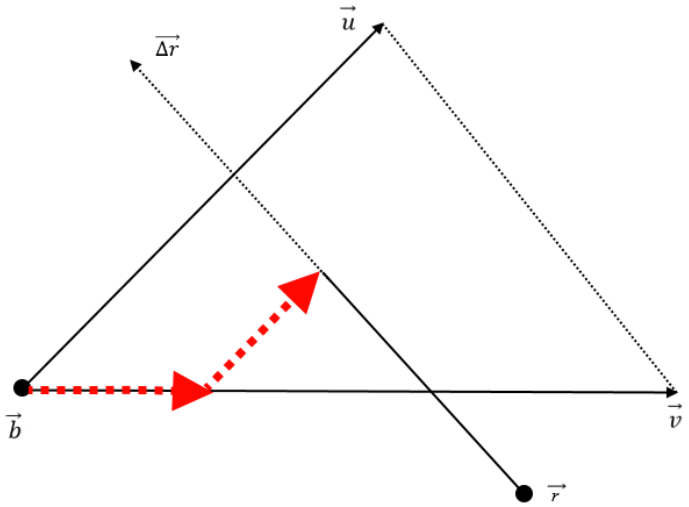

Figure 6. Intersection of a line segment and a triangle in R.[3] Triangle is encoded as position of one vertex $\vec{b}$ and relative positions of two others $\vec{u}$ and $\vec{v}$. Line segment is encoded as starting position $\vec{r}$ and relative position of end $\overrightarrow{\Delta r}$.

Here this vector is denoted as $\vec{b}$ where b stands for boundary position. Similarly other vertices can also be described with vectors of their position, but instead of representing them as vectors of their position with respect to origin we represented them as their position with respect to the first vertex. These two vectors are named $\vec{u}$ and $\vec{v}$ (boundary u and v vectors).

The path of travel is represented as a line segment. A line segment in an arbitrary position in space can be represented as a set of two vectors: one for where it begins and one for where it ends with respect to the first one. Here the starting position is called $\vec{r}$ and ending position with respect to starting position is called $\overrightarrow{\Delta r}$. The condition for a collision can be determined by solving the following vector equation

$$\vec{r} + k_0\overrightarrow{\Delta r} = \vec{b} + k_1\vec{u} + k_2\vec{v} \quad , \tag{12}$$

whose solution gives

$$k_0\mathfrak{D} = b_2u_1v_0 - r_2u_1v_0 - b_1u_2v_0 + r_1u_2v_0 - b_2u_0v_1 + r_2u_0v_1 + b_0u_2v_1 - r_0u_2v_1 + b_1u_0v_2 - r_1u_0v_2 - b_0u_1v_2 + r_0u_1v_2 \tag{13}$$

$$k_1\mathfrak{D} = b_2\Delta r_1v_0 - b_1\Delta r_2v_0 + \Delta r_2r_1v_0 - \Delta r_1r_2v_0 - b_2\Delta r_0v_1 + b_0\Delta r_2v_1 - \Delta r_2r_0v_1 + \Delta r_0r_2v_1 + b_1\Delta r_0v_2 - b_0\Delta r_1v_2 + \Delta r_1r_0v_2 - \Delta r_0r_1v_2 \tag{14}$$

$$k_2\mathfrak{D} = -b_2\Delta r_1u_0 + b_1\Delta r_2u_0 - \Delta r_2r_1u_0 + \Delta r_1r_2u_0 + b_2\Delta r_0u_1 - b_0\Delta r_2u_1 + \Delta r_2r_0u_1 - \Delta r_0r_2u_1 - b_1\Delta r_0u_2 + b_0\Delta r_1u_2 - \Delta r_1r_0u_2 + \Delta r_0r_1u_2 \quad . \tag{15}$$

Here

$$\mathfrak{D} = \Delta r_2u_1v_0 - \Delta r_1u_2v_0 - \Delta r_2u_0v_1 + \Delta r_0u_2v_1 + \Delta r_1u_0v_2 - \Delta r_0u_1v_2. \tag{16}$$

These k_i represent what fraction of each vector is sufficient to make the intersection happen. Note that 0 based indexation is used in matrix components ($\langle r_0, r_1, r_2\rangle = \langle x, y, z\rangle$). We know that an intersection happens if and only if all of the following conditions are met

i. $0 \le k_0 < 1$ *(17)*

ii. $0 \le k_1$ *(18)*

iii. $0 \le k_2$ *(19)*

iv. $k_1 + k_2 \le 1.$ *(20)*

The first condition is defines a line segment and conditions ii, iii and iv defines a triangle. When a collision is detected using these criteria, the particle is advanced to the point of collision with the boundary, $\vec{r} + k_0\overrightarrow{\Delta r}$, and the unused portion of the kick is reflected of the boundary. As seen in the sketch in Figure 7, reflection of an incident vector with respect to a surface normal is similar to sides of an equilateral parallelogram.

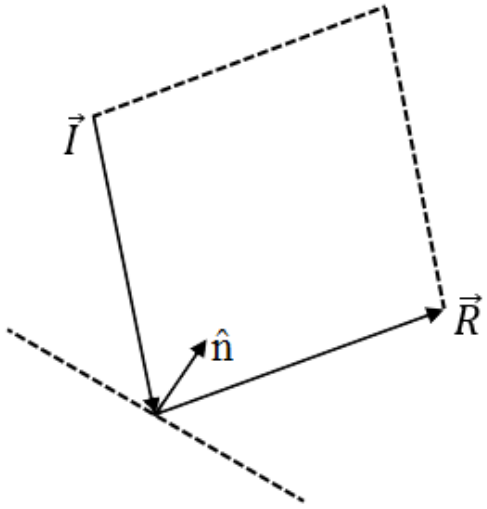

Figure 7. Reflection of a line segment from a surface. $\vec{I}$ is the incident vector. $\vec{R}$ is the reflected vector. $\hat{n}$ is the surface normal.

The reflected vector is the sum of incident vector and long diagonal vector. To calculate the long diagonal, first the component of incident in the direction of surface normal has to be calculated. Assuming that the surface normal is normalized this component is $(-\vec{\boldsymbol{I}} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$. This is half of the long diagonal. When multiplied by two and summed with the incident vector, the reflected vector becomes

$$\vec{R} = \vec{I} - 2(\vec{I} \cdot \hat{n})\hat{n} \,. \tag{21}$$

These equations are used to simulate the interaction between particles and boundaries.

2.3 Optical Model

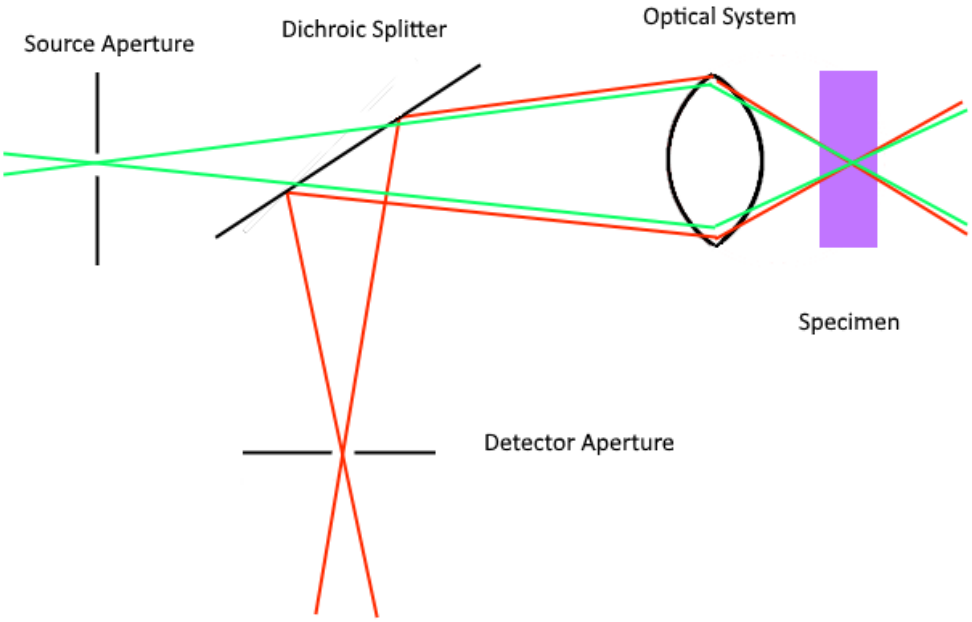

Figure 8. Schematic showing the basic principles of a confocal micropcope.

Fluorescent molecules emit light in every direction equally. When a photo-detector is used to measure brightness of fluorescent molecules, the brightness data that it detects includes significant amount of light from above and below the focal plane. The FRAP process described in this project uses confocal imaging to get the brightness data. Confocal imaging refers to a type of fluorescence microscopy [6]. It is a noise reducing technique that is used to get data from a specific cross section of the specimen. A source laser shines through the aperture and optical system to the specimen, where it excites fluorescent molecules. Fluorescent molecules emit light which goes back through the system to the detector (as illustrated in Figure 8).

Let us consider a specimen that is assumed to be at the right and a detector at the left. This method uses a combination of optical lenses aligned with a pinhole aperture. The opening of this aperture is located at the left focal point (for emission wavelength) of the lens. Since we assume that in a well-designed microscope there is no exterior light leaking in, this mostly only allows light from sources located at the right focal pass to the detector. This creates a “focal plane”. This focal plane is the plane defined parallel to the lens and the aperture. Note that this will later be called *z-axis*

while describing the Gaussian Beam. At the equivalent point there is a light source-aperture pair. This equivalent point is usually achieved by using a dichroic beam splitter, since incident and returned light have different wavelengths. This is used to illuminate fluorescent molecules which emits light back. The technique simulated is known as Scanning Confocal Microscopy because each pixel in the domain is scanned sequentially by the microscope as opposed to capturing the entire image at the same time.

There are two types of light sources in our model: the laser and fluorescent particles. Both are assumed to behave as Gaussian beams because Gaussian Beam model fits the experimental data well [3]. Such a Gaussian beam is given by

$$I = I_0 \left(\frac{w_0}{w(z)}\right)^2 \exp\left(\frac{-2r^2}{w^2(z)}\right). \quad (22)$$

Here I is the intensity, I_0 is the intensity at the center of the beam, w_0 is the standard deviation of the intensity with respect to r-axis when $z = 0$, and is an input parameter. r is the horizontal distance from the focal point to the particle, and $w(z)$ is given by

$$w(z) = w_0 \sqrt{1 + \left(\frac{z}{z_r}\right)^2}, \quad (23)$$

where the Rayleigh range, z_r, is an input parameter, it denotes the z distance it takes for variance to double.

To avoid using a square root function, our computational implementation of this function performs the calculation differently. A new variable called w_n is defined that has a value of $\left(\frac{w}{w0}\right)^2$, which yields

$$I = \frac{I_0}{w_n} \exp\left(\frac{-2(x^2 + y^2)}{w_n w_0^2}\right) \quad . \quad (24)$$

for the total intensity I. In the following, we discuss how this beam is used for imaging.

2.3.1 Imaging

The light source emits a Gaussian Beam with respect to the particle and the particle emits a Gaussian Beam with respect to the detector. Therefore I_0 has to be multiplied twice with

$$\frac{I}{I_0} = \frac{1}{w_n} \exp\left(\frac{-2(x^2 + y^2)}{w_n w_0^2}\right) \quad (25)$$

to get the correct intensity detected at a given pixel location. The brightness of each pixel is the sum of the values computed by this function for each of the particles.

2.3.2 Bleaching

The bleaching and imaging steps may use different beam parameters (w_0 and z_r). The bleaching beam is not squared because for the bleaching process, the laser light source is unidirectional. In FRAP the bleaching is done by shining high intensity light on specific are of the cell, which is for our work a circle. The circle shape is achieved by shining small Gaussian beams in the circle area

sequentially (scanning). For our system, horizontal movement of the microscope is assumed to take negligible time (1.4 μs) compared to vertical movement (360 μs) therefore, a line of bleach is simulated as a single pulse [3]. This is achieved by convolving Gaussian across a horizontal line. The beam intensity then becomes

$$I_{bleach}(x,y,z) = \frac{I_0}{\sqrt{w_n}} \frac{-\left(\mathrm{erf}\left(\frac{x-a}{w_n w_0^2}\right) + \mathrm{erf}\left(\frac{b-x}{w_n w_0^2}\right)\right)}{2} \exp\left(\frac{-2(y^2)}{w_n w_0^2}\right). \quad (26)$$

The probability of a particle being bleached is a monotonically increasing function of intensity. For low power, this relation can be approximated as binomial probability (c_1,c_2 and c_3 are arbitrary)

$$P(I_{bleach}) = c_1(1 - e^{-c_2 I}) \quad , \quad (27)$$

which has a Taylor approximation

$$P(I_{bleach}) = c_3 I + O(I^2) \quad . \quad (28)$$

This is valid in the limit of weak photobleaching. Therefore, the expression for bleach probability becomes

$$P_{bleach}(x,y,z) = \frac{P_0}{\sqrt{w_n}} \frac{\left(\mathrm{erf}\left(\frac{x-a}{w_n w_0^2}\right) + \mathrm{erf}\left(\frac{b-x}{w_n w_0^2}\right)\right)}{2} \exp\left(\frac{-2(y^2)}{w_n w_0^2}\right). \quad (29)$$

In this case, P_0 is the probability a particle will be bleached at the center of the beam. A random number from 0 to 1 is generated, and if the number is smaller than the value of Eq. (29) at that point, then the particle is bleached and will not be included in image any more.

The function erf in Eq. (29) had to be manually implemented and for that we used the approximation [7]

$$\operatorname{erf}(x) \approx \begin{cases} 1 - \dfrac{1}{(1 + 0.278393x + 0.230389x^2 + 0.000972x^3 + 0.078108x^4)^4} & if\ x \geq 0 \\ -\operatorname{erf}(-x) & otherwise \end{cases} \quad (30)$$

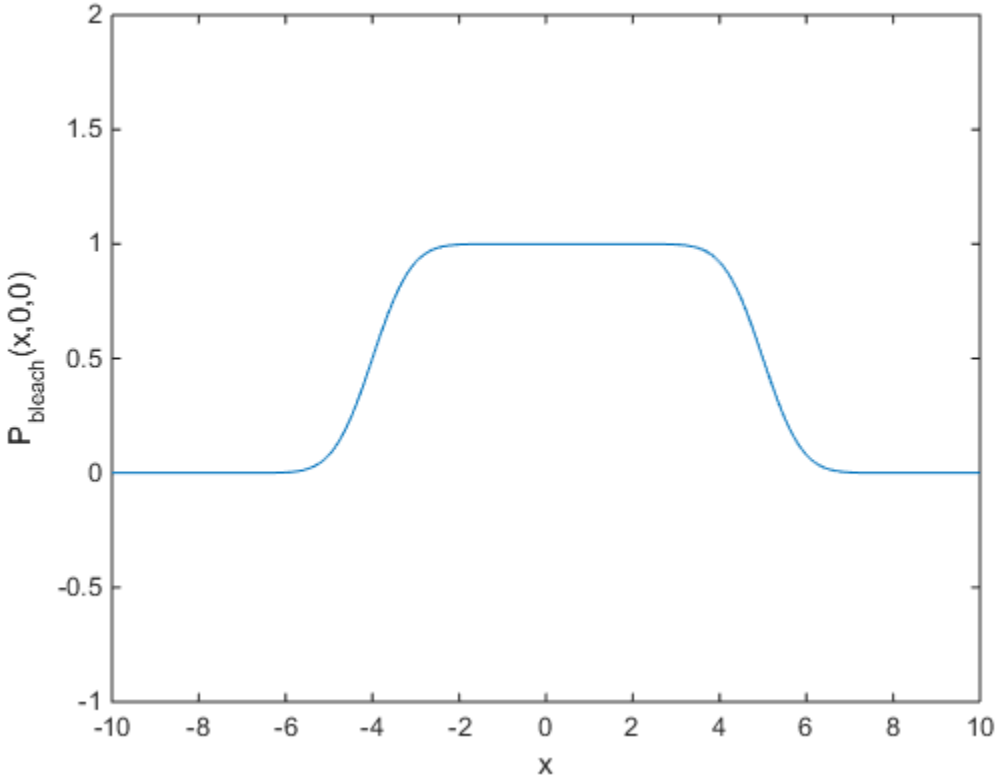

Figure 9. Bleach probability $P_{bleach}(x,0,0)$ for a=-4, b=5, P_0=1 and w_0=1.

In physical model, Gaussian beam is assumed for optical properties and Brownian motion is used to compute the diffusion.

2.4 Computational Implementation

2.4.1 External Libraries

This code makes extensive use of external open-source java libraries. One of the key elements that make this code fast given the amount of computation it has to perform, is the utilization of GPUs. A significant portion of the simulation is highly parallelizable. Aparapi is a java library that converts java code to OpenCL, to be executed on different computational units including GPUs and multi-processor CPUs. In this project, we used aparapi to do GPU executions [8].

The other external library used is the PRNGKernel library. PRNGKernel is an open-source random number generator library written for aparapi. The current version includes XOR shift and Mersenne Twister as the random number generator algorithm options. Here we used the XOR shift due to its long period and simplicity.

2.4.2 Mesh Input

STL is a surface mesh format that describes a surface as a set of triangles as seen in Figure 10. A triangle is encoded as a set of nine 32-bit IEEE floats: <x,y,z> for each of the three vertices [9]. This file type has two standard encodings: ASCII and binary. ASCII is more human readable and easier to implement, whereas binary is more compact, more common and harder to implement. Our approach assumes binary STL format as the type of the Mesh File. Inputting this format is a not trivial task. Binary STL uses little endian for multi-byte number encoding. The architectures we worked on use big endian. These little endian numbers were converted to big endian numbers by a series of left shifts. Binary STL starts with an 80-byte header, and these are skipped. The next number is the number of mesh elements. This number is used to preallocate memory. Then for each triangle there are four vectors of three floats: coordinates and surface normal. While inputting the mesh file, the minimum and the maximum of each coordinate is stored in a separate variable which is used for image positioning, bleach positioning and particle initialization. Unlike STL, we encoded triangles as position of one vertex and relative positions of two others, rather than storing position from the origin. Please see Appendix 3 for STL input details.

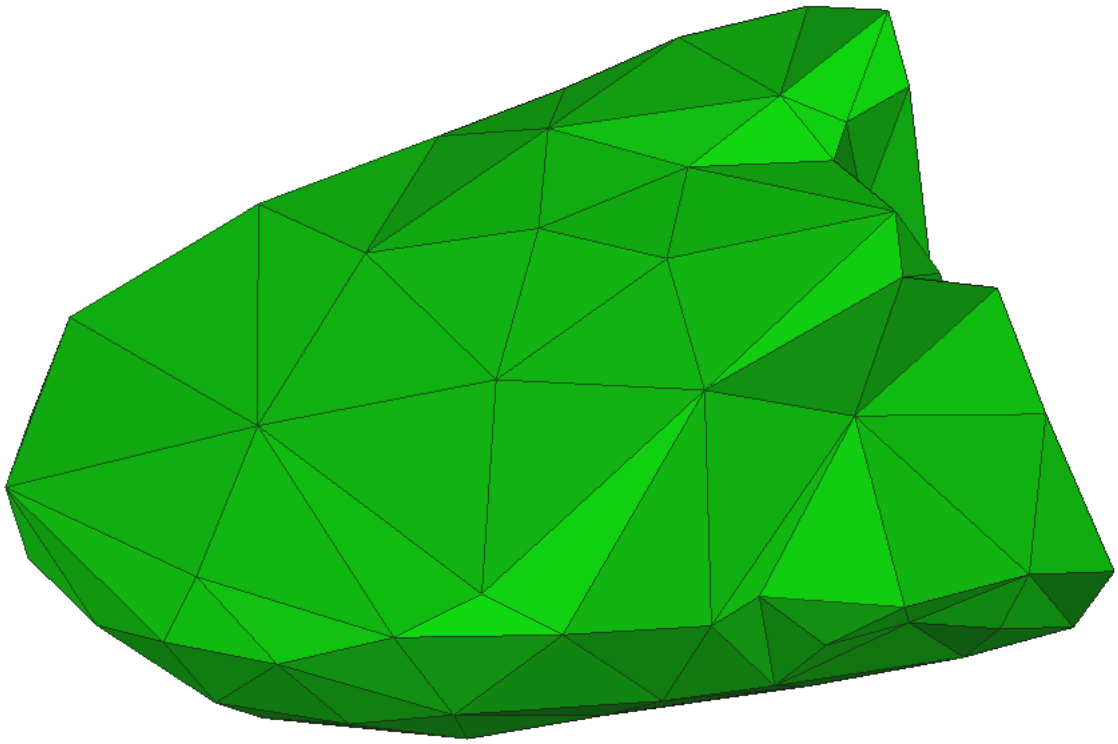

Figure 10. STL surface as a set of triangles generated from experimental images.

2.4.3 The Particle Initialization

After the mesh has been input, particles are initialized. This is done by picking a random point inside a rectangular prism defined by the maximum of each coordinate. Then, if this particle is inside the mesh surface boundary it is kept in position. If not, the position is randomly assigned again until the particle is inside the boundary.

Deciding whether a point is inside the boundary is done by using the triangle-line segment intersection algorithm mentioned above. Taking a very large line segment (twice the size of the prism) starting at the position of the particle and counting the number of mesh elements the line segment intersects with, can be used to determine which side of the boundary point is in. If the number of intersections is odd, the point is inside the boundary. If it is even, the point is outside the boundary.

2.4.4 Event Sequencing

The sequence of events is input from the file "FrapSettings.txt". As described in Appendix 1: Recipe, the syntax of the sequence part of the settings file is "[Event Type] [Number of Repetitions] [Time between Two Repeated Instances]". The input function first checks for consistency and then puts the events in a linked list. A linked list was preferred because it is easy to add an item and iterate through the list in Java.

2.4.5 Image Output

The designed program assumes that the imaging for a given geometry is done from top to bottom with respect to y-axis. User is expected to input the mesh file this way. Whenever the counter for imaging reaches the last y-row, a method (function) is invoked to output the density matrix as a PNG image. In each instance of the program, the brightest pixel of the first image is used to normalize the output to 15 bits. java.awt.image.BufferedImage is a standard class in Java that is used for image input and output. The instances can be set to be of desired encoding such as ARGB, RGB, 8 bit grayscale and 16 bit grayscale. However, not all of the implemented functions work with all types, therefore we had to manually implement the value setting function. We already had a matrix that holds the values of the intensity of a certain pixel. We used the WritableRaster class in order to manually set each of the pixels for the BufferedImage.

2.4.6 Summary

The simulation starts by inputting the mesh, settings and events. The events are executed on GPU and image files are outputted as PNG. See Figure 11 for the flowchart.

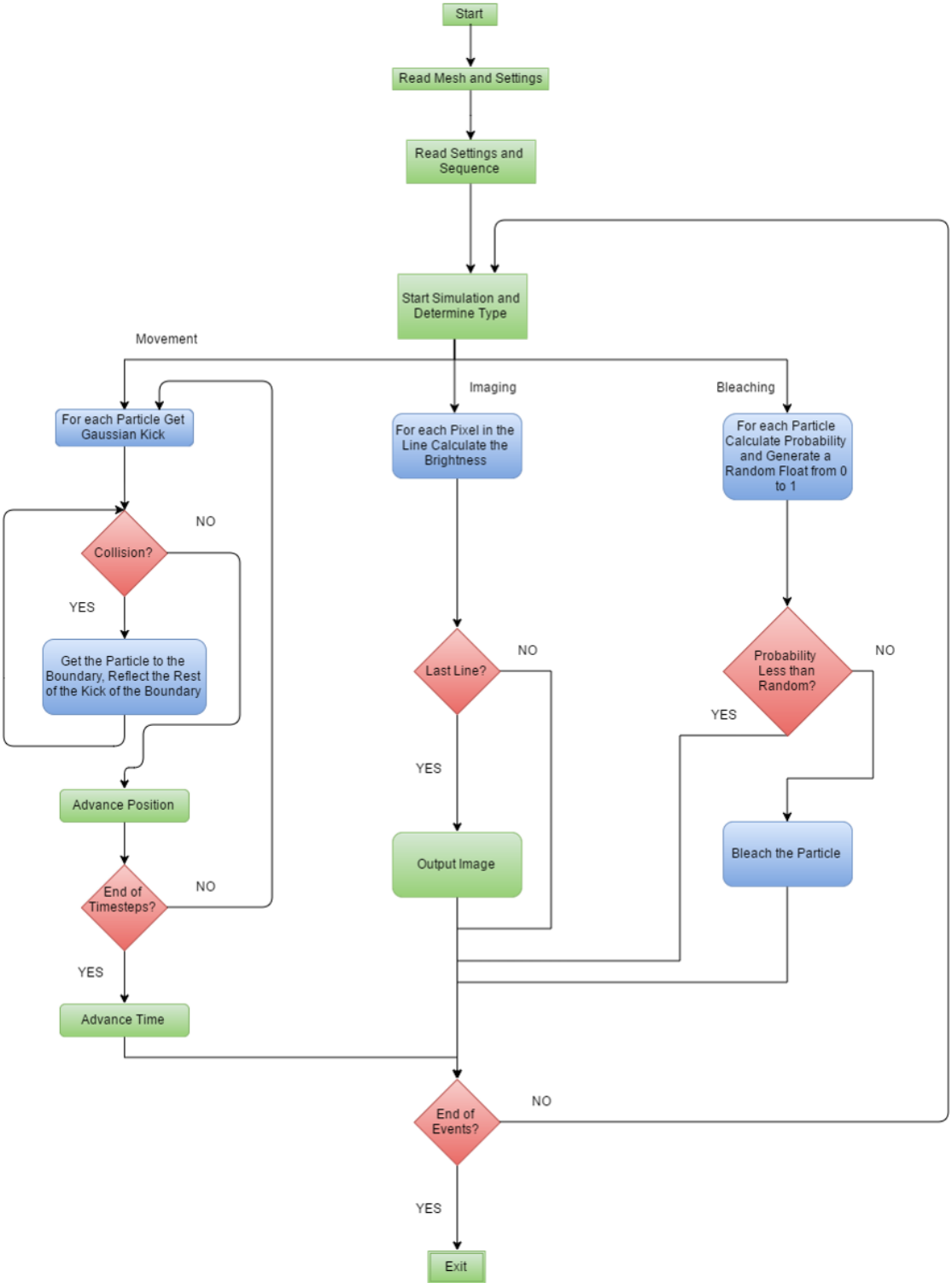

***Figure 11**. Flow chart of the implementation.*

3 VALIDATION AND BENCHMARKS

3.1 VALIDATION

It is important to be able to demonstrate that our computational implementation of FRAP works. There are few ways in which this can be done. For instance, the Mean Squared Displacement (MSD) of particles can be calculated and compared to expected diffusion behavior [2]. Comparing the simulated recovery curves with known recovery curves is another way to assess the accuracy of the implementation.

3.1.1 Comparison to Open Boundary Solution

In order to compare with the analytical solution for open boundaries, we performed simulations. To get a baseline, we imaged 10 times over the first 10 seconds. Next, a double bleach was applied to ensure the ROI is completely dark. Finally, the domain was imaged for 16 seconds. The program can use a different base intensity for each of the program instances for visibility purposes. The intensity of the ROI is averaged for the initial 10 images. This yields a scale factor to divide each of the data points by for normalization. The time of the first image after bleach is subtracted from the time axis in the plots for easier readability.

As discussed in Section 1.3 earlier, Eq. (3) represents the recovery curve in an open boundary, i.e.,

$$f(t) = 1 - \phi + (\phi e^{-\alpha})\big(I_0(\alpha) + I_1(\alpha)\big).$$

Since we know fluorescent molecule distribution follows the uniform diffusion equation in time, running a simulation that is quantitatively close to homogenous in two dimensional open domain should yield a very similar recovery curve. For this purpose, we used a very flat cylinder that is significantly larger than expected mean square displacement in unit time to minimize the boundary effect ($D = 1\ \mu m^2/s$, $R_{boundary} = 10\mu m$). The domain used in these simulations is shown in Figure 12.

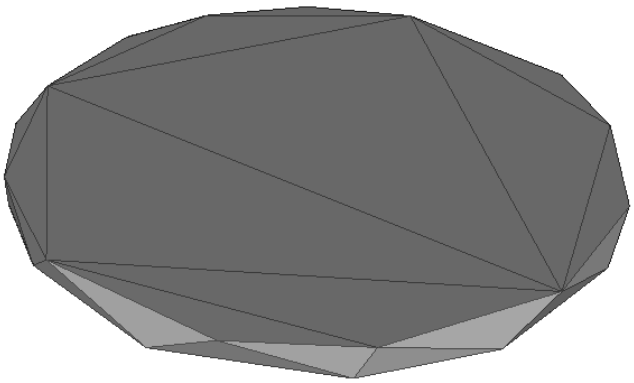

Figure 12*. Mesh used for the flat cylinder.*

The Rayleigh Range, z_r, values for both bleaching and imaging beams were very high (10000 μm) to create a two dimensional effect. Beam with, w_0, values are very small for both beams (0.0223 μm). Beam widths are small relative to the ROI to decrease image blur and to create a sharper

bleach profile. The configuration file used in the simulations can be found in Appendix 4. A sample Point Spread Function (PSF) is shown in Figure 13.

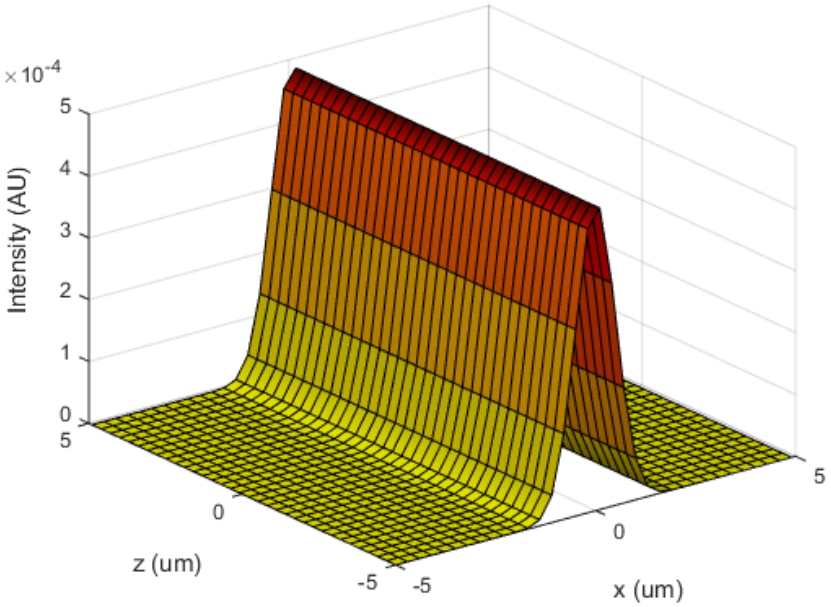

Figure 13. *Gaussian beam used (z_r = 10000 μm, w_0 = 0.0223 μm).*

Fitting the simulation data to the open boundary solution with the diffusion coefficient as a free parameter results in an excellent fit shown in Figure 14,

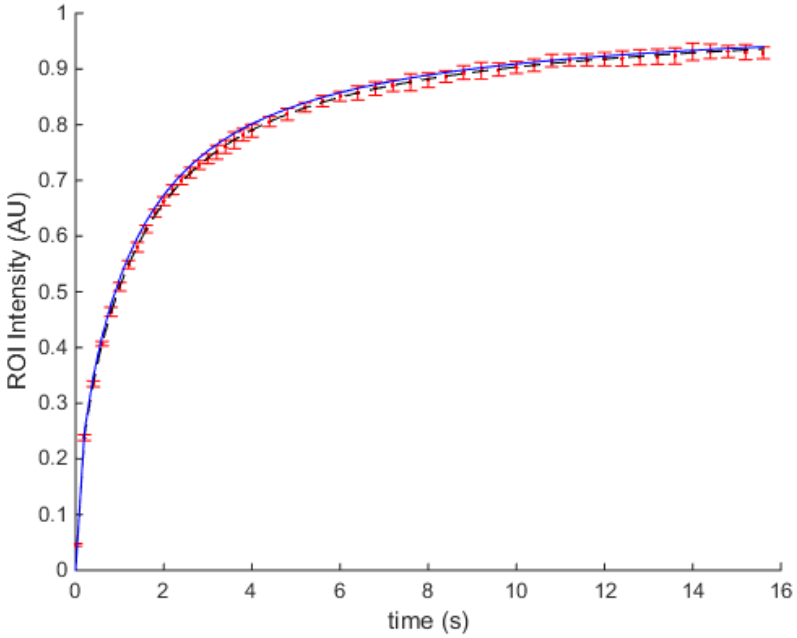

Figure 14*. Recovery curve for a flat cylinder. Dotted error bars are output data, solid line is open boundary solution with input parameters, dashed is the fitted solution where Φ=1.*

with $D_{calculated} = 0.94\ \mu m^2\ /\ s$. Approximately 7% error is likely to be caused by the PSF. Therefore, we made another branch of the program that only bleaches everthing inside ROI (no PSF) and outputs the number of paticles inside the ROI. This gives a very close approximation to the analytical solution $D_{calculated} = 0.99\ \mu m^2/s$.

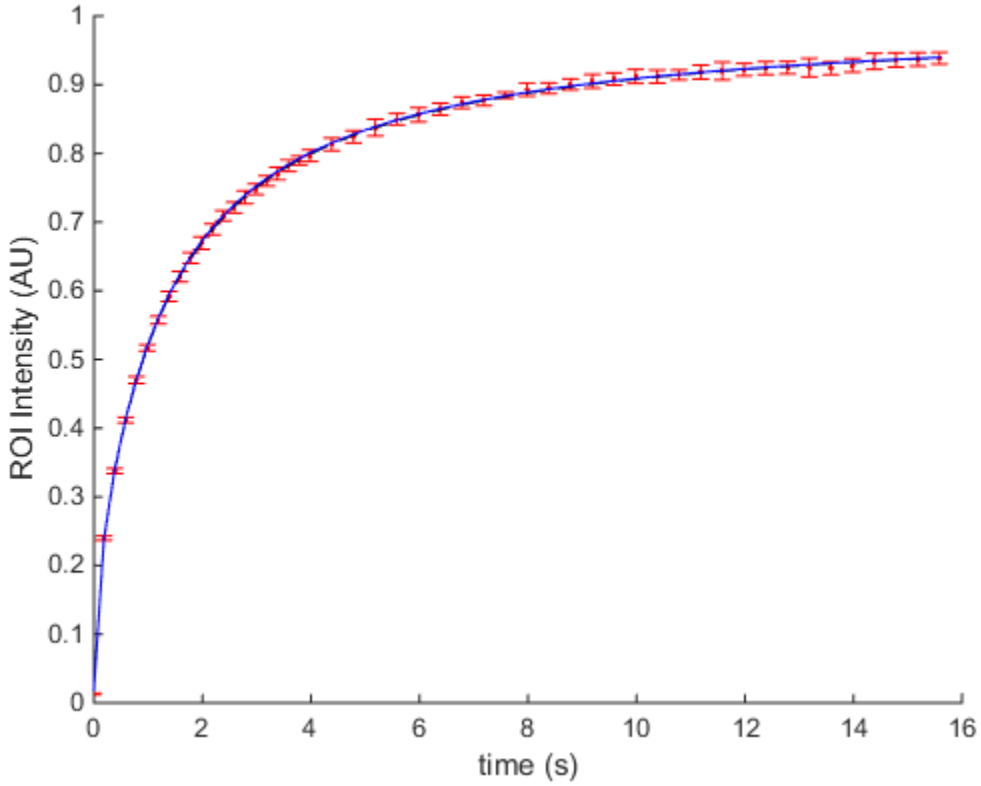

Figure 15*. Recovery in flat cylinder with no PSF. Points with error bars are output data, solid line is open boundary solution with input parameters, dashed is the fitted solution where Φ=1. Fitted solution is almost invisible because it is masked by open boundary solution.*

Since the data generated by the simulations and the analytical solution are very close, we conclude that the simulation is working accurately. The slight underestimation of the calculated *D* can be attributed to finite size effects.

3.1.2 Comparison to a Known Working Model

We also compared our simulations with the results in Ref. [3]. The recovery curve for the parameters given in appendix X is shown in Figure 16 with the data points. We repeated these simulations using our package but with simplified bullet like geometry in Ref. [3], and the results are shown in Figure 16 with a solid line. The results show excellent agreement between two simulation approaches.

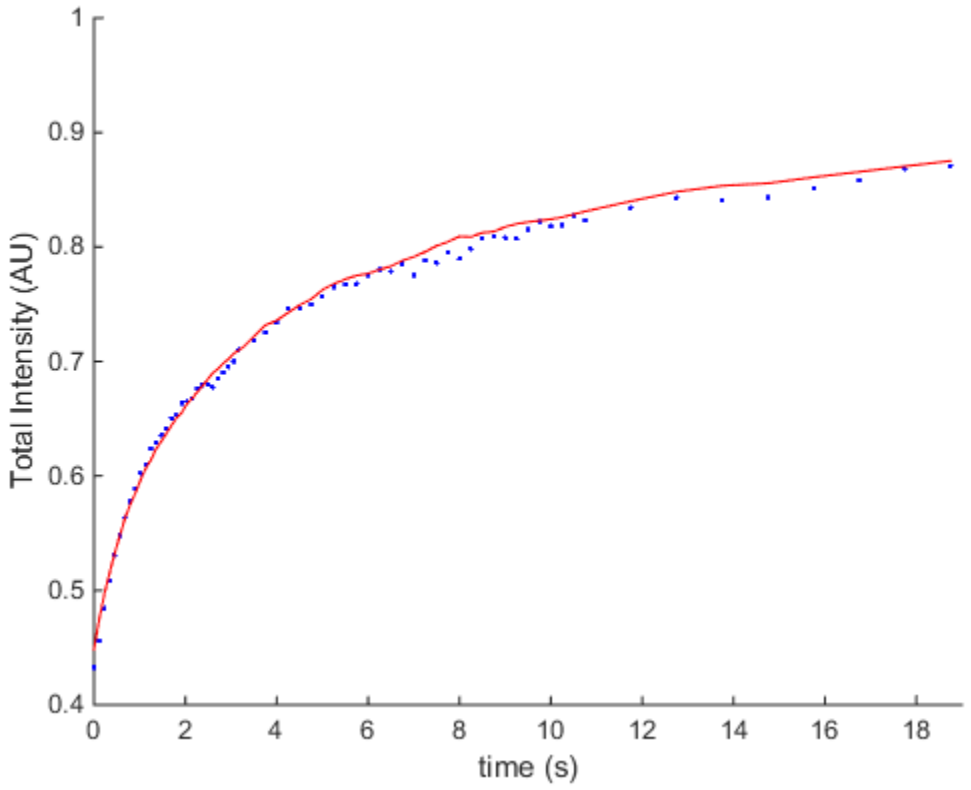

***Figure 16**. Comparison of recovery curves with our data (solid red line) and Ref. [3] (blue points).*

3.2 TIME BENCHMARKS

It is important to understand time how execution time is affected by the input parameters. This allows users to assess time/accuracy tradeoff while running the program. In what follows, we will present time benchmarks for different parts of the code separately.

3.2.1 Movement

Movement kernel checks for collisions. If there are collisions, the kernel performs the collision event described above, if there are no collisions, increments the coordinates. It is expected that the time it takes to execute the movement will depend on the number of mesh surfaces, the number of particles and time steps. Figures 17 through 22 show the movement execution time as a function of these variables and movement execution time per increment of variable as a function of these variables.

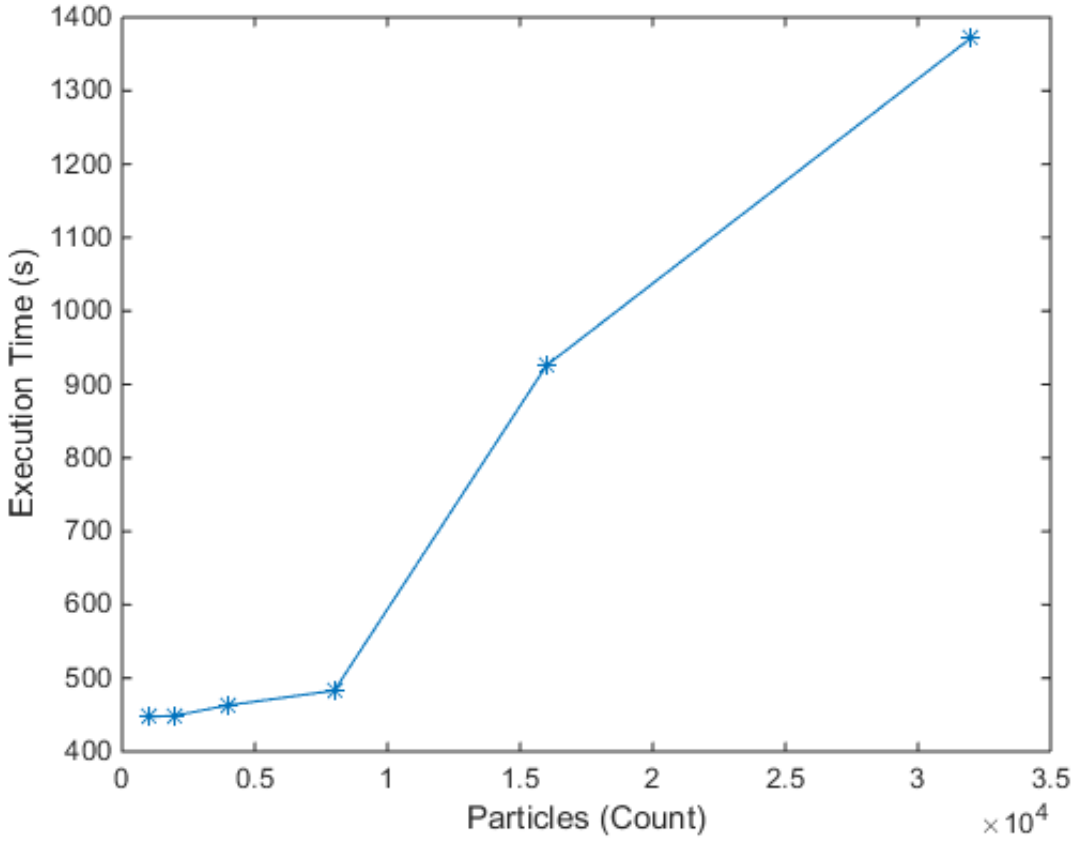

***Figure 17**. Execution time of movement as a function of number of particles. Simulation time = 2.56 s.*

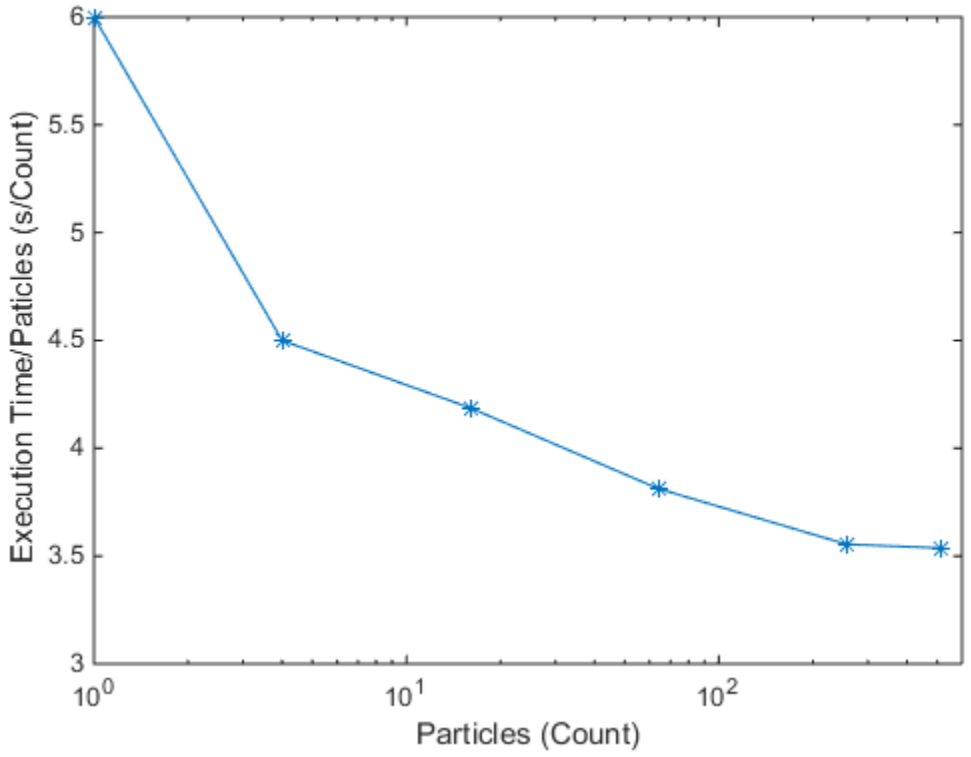

Figure 18. *Execution time of movement per particle as a function of number of particles. Simulation time = 2.56 s.*

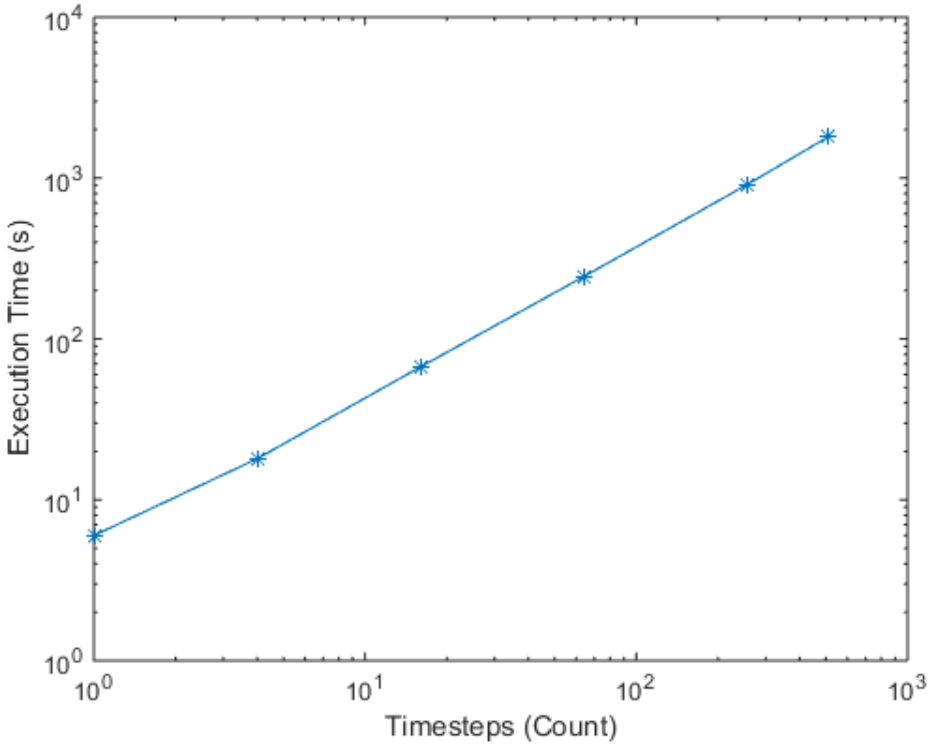

***Figure 19**. Execution time of movement as a function of number of time steps. Note that the time step duration does not change, simulation time increases as the number of time steps increase. Particles= 8000.*

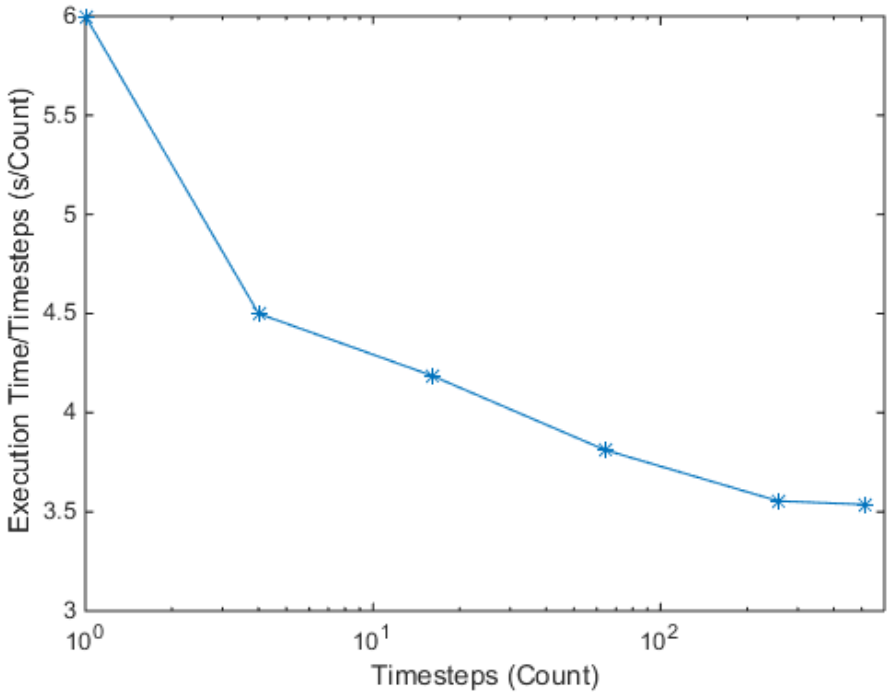

***Figure 20**. Execution time of movement per time step as a function of number of time steps. Note that the time step duration does not change, simulation time increases as the number of time steps increase. Particles= 8000.*

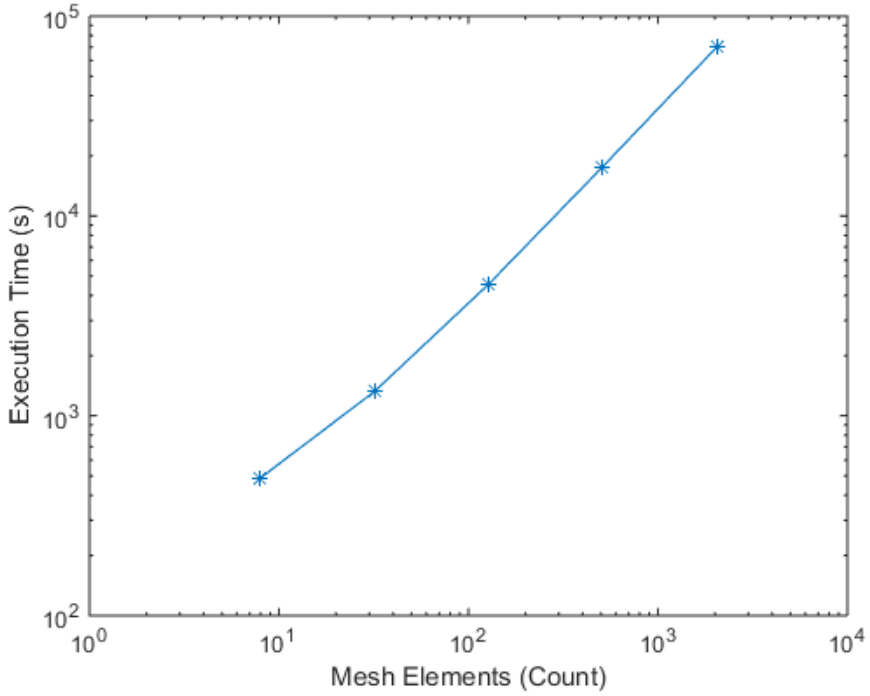

***Figure 21**. Execution time of movement as a function of number of time steps. Particles= 15000, Simulation time = 2.56 s.*

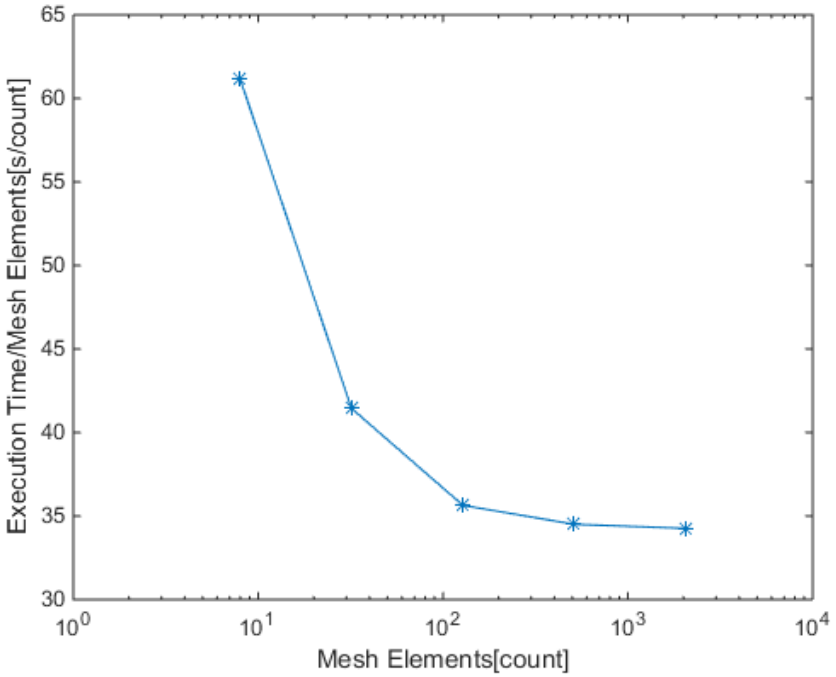

***Figure 22**. Execution time of movement per mesh element as a function of number of mesh elements. Particles= 15000, Simulation time = 2.56 s.*

The scaling with respect to all three of these parameters is linear as expected.

3.2.2 Imaging

Similarly, the imaging execution time is expected to be dependent on the number of pixels and number of particles. Simulations are performed with w_0=0.5 μm and z_r=0.7 μm. and the results are shown in Figures 23 through 26.

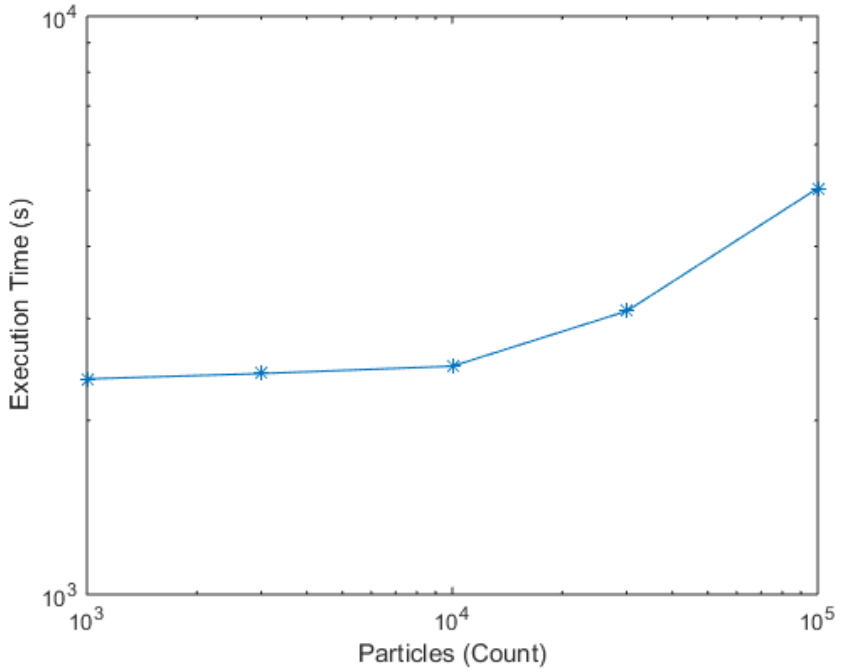

Figure 23*. Execution time of imaging as a function of number of particles. 400 pixels in x direction.*

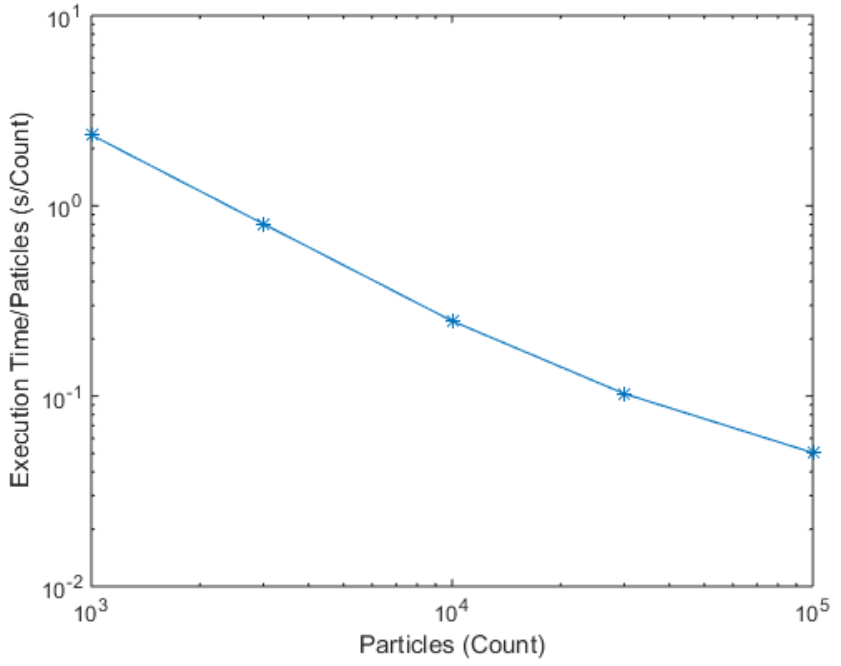

Figure 24*. Execution time of imaging per particle as a function of number of particles. 400 pixels in x direction.*

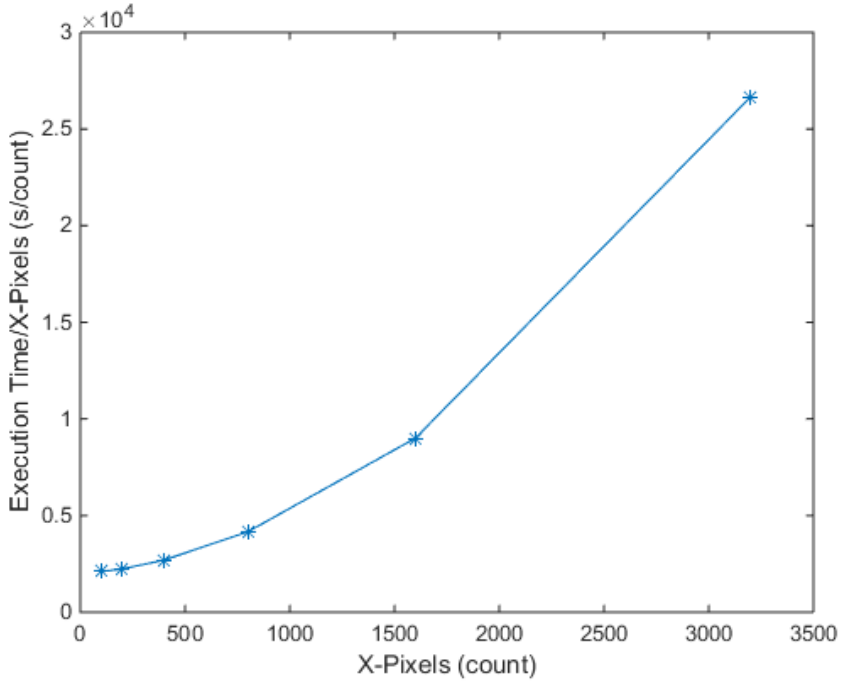

***Figure 25**. Execution time of imaging as a function of the number of pixels in the X direction. Note that Y pixels are scaled proportionally as the number of X-pixels increase. 16000 particles.*

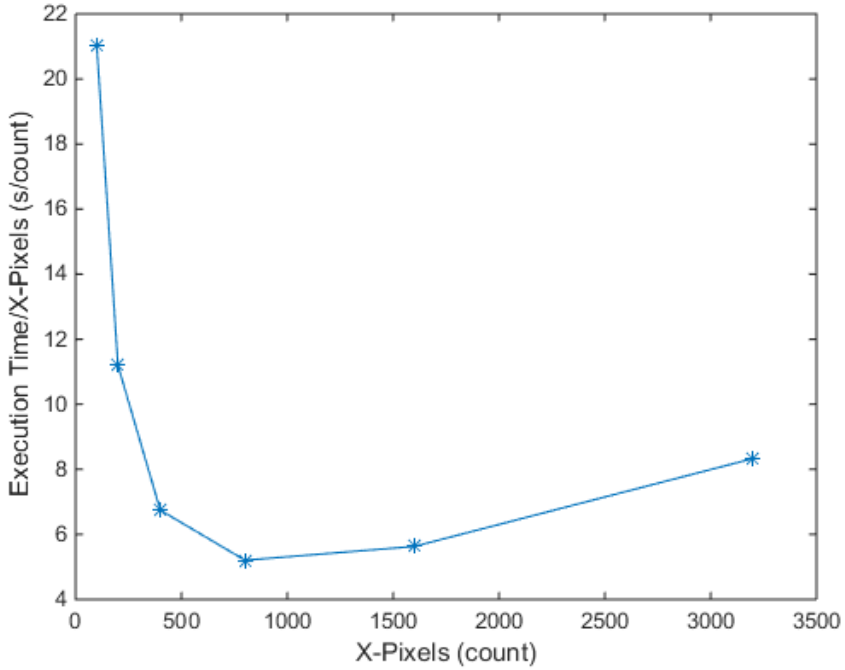

***Figure 26**. Execution time of imaging as a function of number of pixels in the X direction. Note that Y pixels are scaled proportionally as the number of X-pixels increase. This results in the quadratic scaling observed. 16000 particles.*

The scaling is linear with respect to number of particles and quadratic with respect to number of pixels in one direction because the total number of pixels increases quadratically.

3.2.3 Bleaching

The execution time of each bleaching line is expected to be a dependent on the number of particles. The simulation results for w_0=0.223 μm and z_r=10000 μm are shown in Figure 27 and 28.

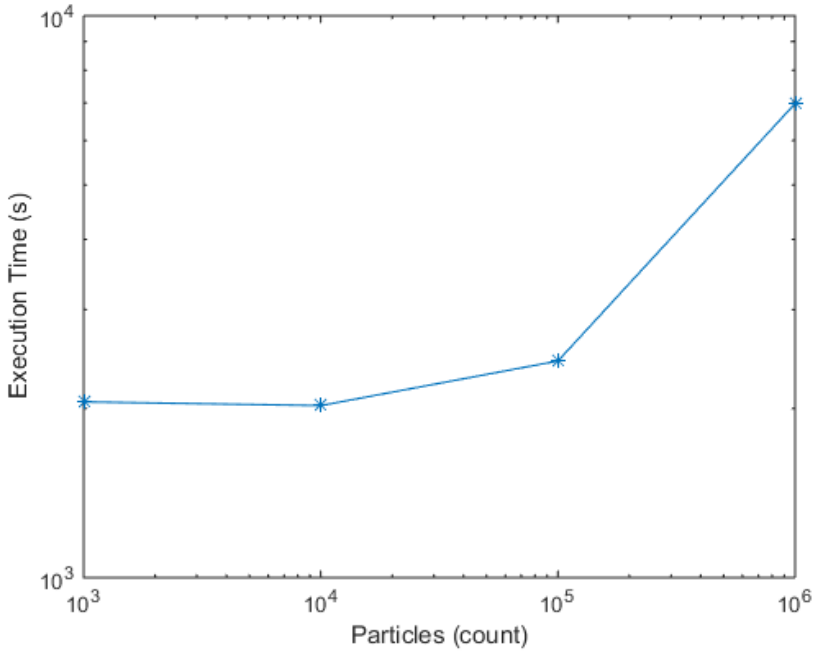

***Figure 27**. Execution time of bleaching as a function of number of particles.*

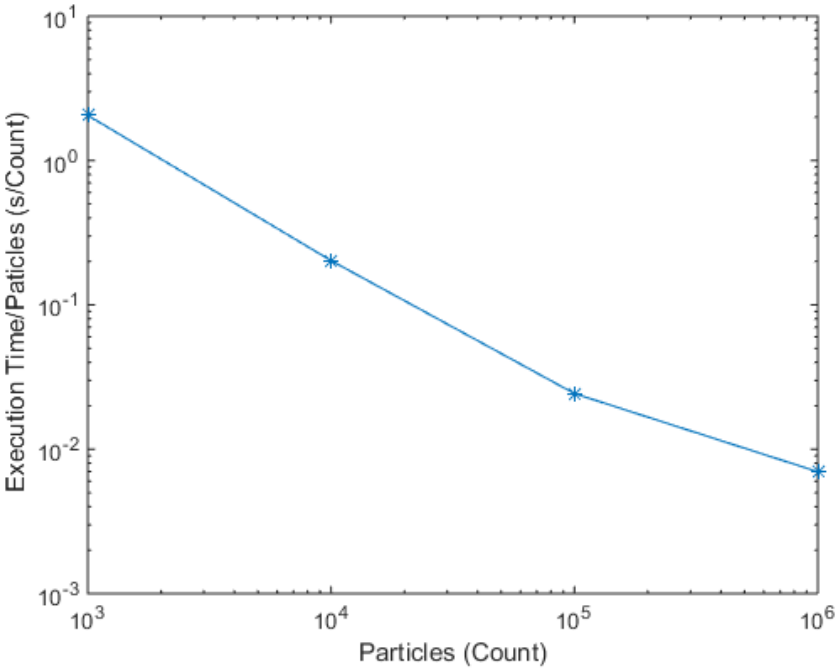

***Figure 28**. Execution time of bleaching per particle as a function of number of particles.*

Bleaching time has linear scaling with respect to number of particles as expected.

3.3 Accuracy Benchmarks

Assessing how recovery curves behave with respect to given input parameters will help decide between accuracy/execution time tradeoff. To do proper estimates, we ran a few instances of the program with same parameters and took the standard deviation at each data point to give us an estimation of the error. This analysis method was used to assess the error with respect to two user inputs, namely the number of particles and the maximum time step size.

3.3.1 Number of Particles

One of the most important parameters to increase the numerical accuracy is the number of particles used in the simulation. The number of particles and numerical error have an inverse relationship. To determine the number of particles required to achieve desired accuracy, the recovery curve for $3X10^3$, $3X10^4$, $3X10^5$ and 10^6 particles were calculated from the simulations (data partially shown in Figure 29). The data sets had 10 simulations each, and the standard deviations are shown as error bars. The settings can be found in Appendix 4.

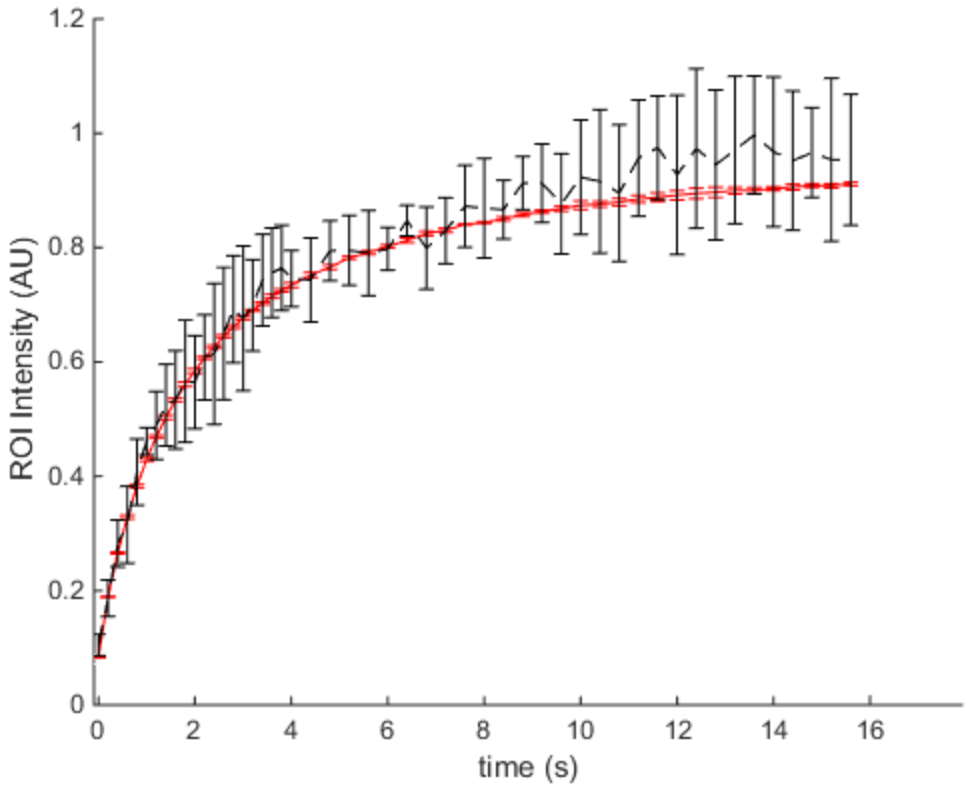

***Figure 29**. Comparison of recovery curves with standard error for $3X10^3$ particles and 10^6 particles. 10^6 particles has almost a negligible standard deviation.*

Since all of these simulations are performed using the same parameters, they have the same data sampling times. Therefore, taking the mean of the standard deviation for a given number of particles shows the relationship between the error and the number of particles (see Figure 30).

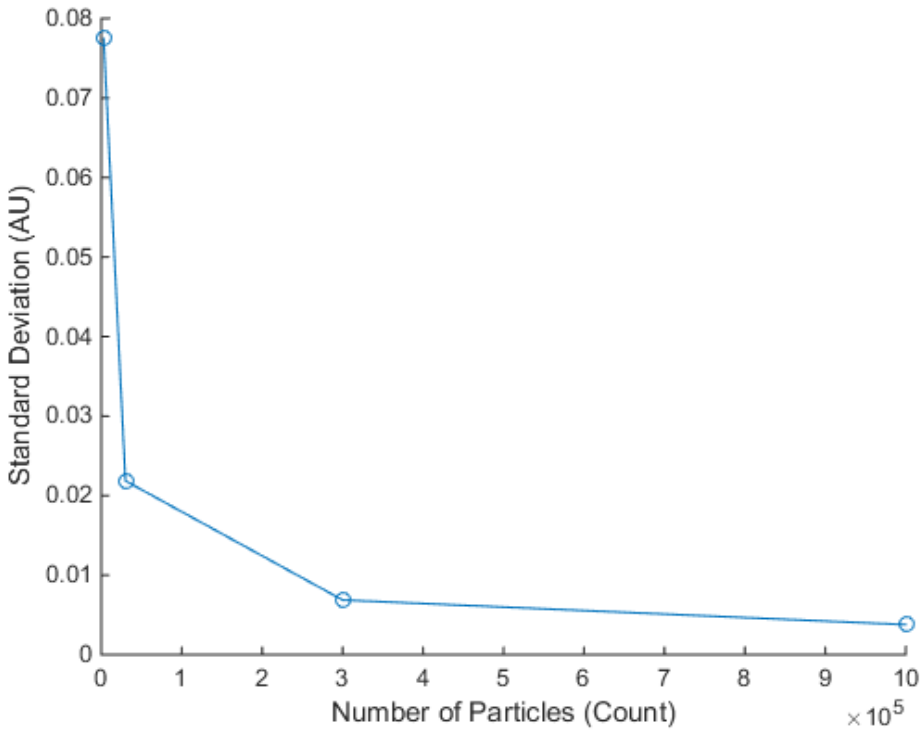

***Figure 30**. Average standard deviation for different number of particles.*

We conclude that using N=1,000,000 provides enough accuracy in the simulations.

3.3.2 Number of Mesh Elements

It is expected that as the number of mesh elements decrease, the shape will be distorted causing the recovery curves to change. Note that the ROI is chosen in the middle of the geometry, and far away from the boundary in this case, to avoid additional complications.

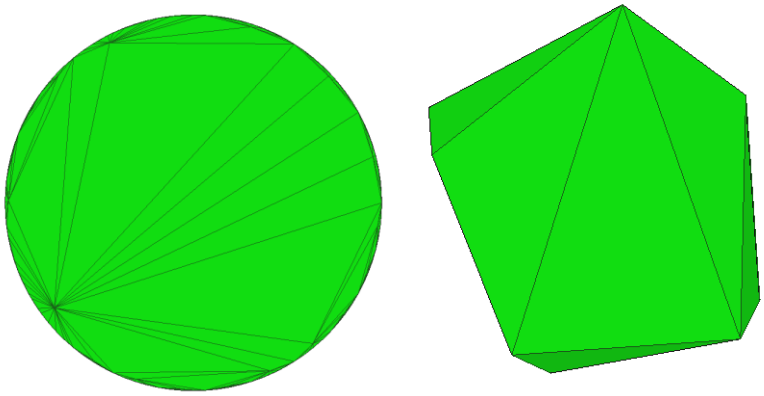

Figure 31. Shape distortion caused by using less triangles (1200 and 12).

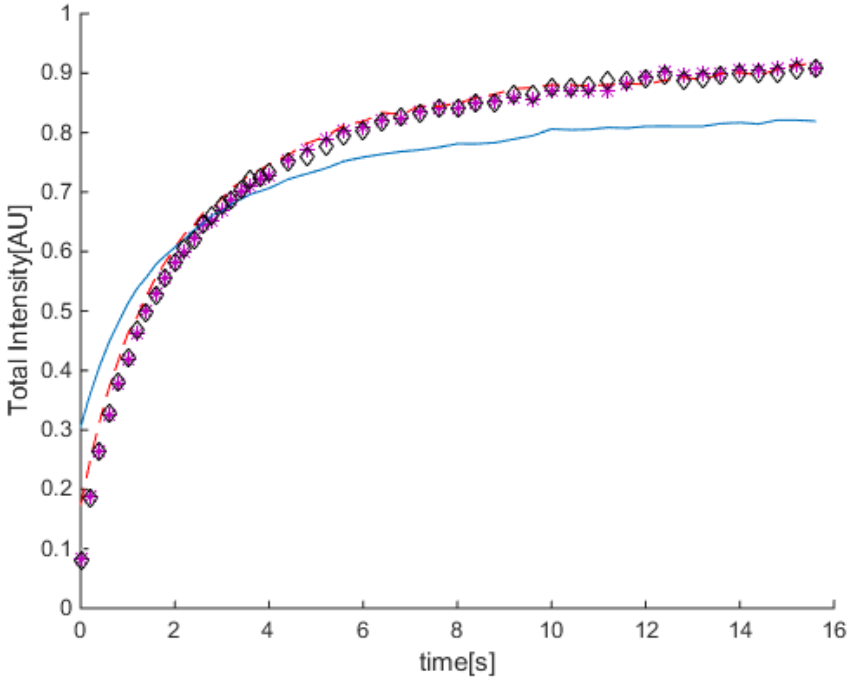

Figure 32. *Comparison of recovery curves inside a geometry approximated by different number of elements. Solid line corresponds to a mesh of 4 elements, dashed line to 12 elements, * to 40 elements, and finally ◇ to 1200 elements on the mesh.*

Computing the absolute errors in each of these cases results in the error plot shown in Figure 32.

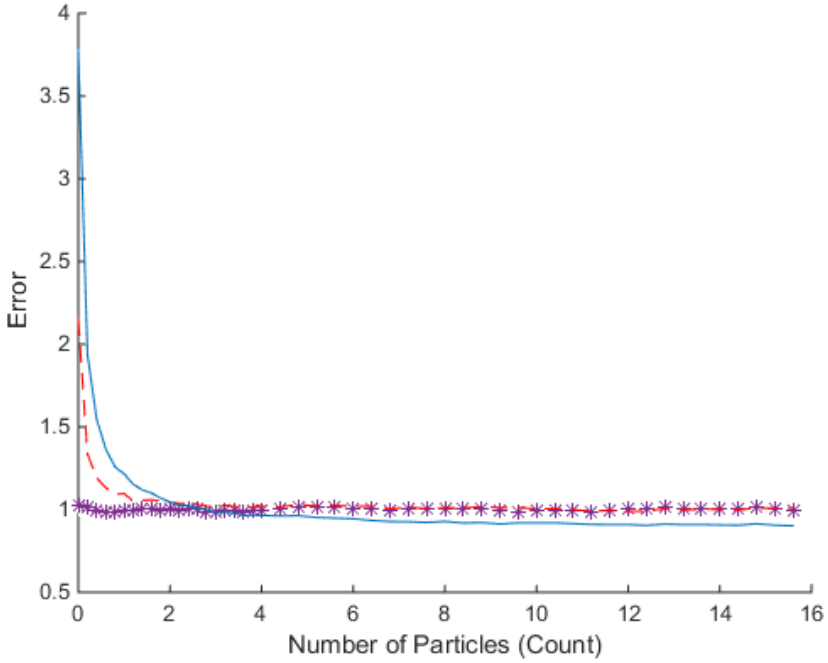

Figure 33*. Comparison of recovery curves divided by recovery curve of 1200 mesh elements inside geometry approximated by different number of elements. Solid line corresponds to a mesh of 4 elements, dashed line to 12 elements, and * to 40 elements on the mesh.*

A similar analysis near the boundary (instead of middle) yields similar convergence (see Figures 34 and 35).

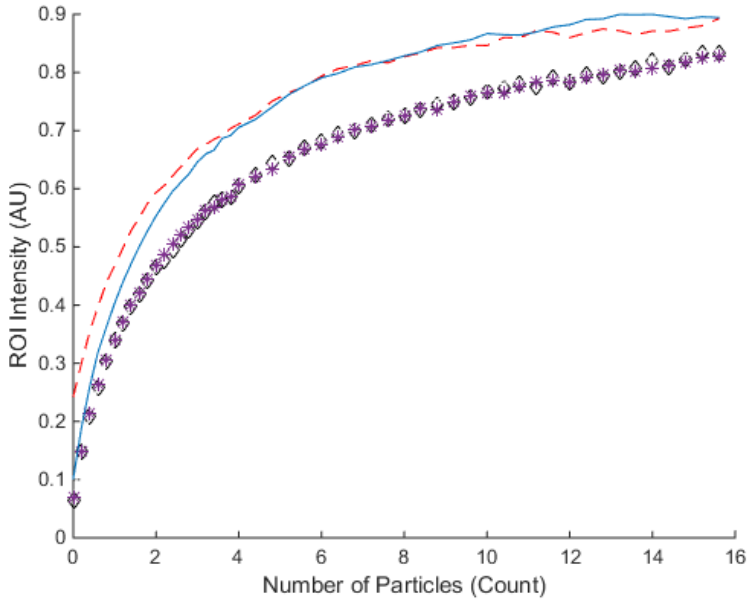

Figure 34*. Comparison of recovery curves near the boundary approximated by different number of elements. Solid line corresponds to a mesh of 4 elements, dashed line to 12 elements, * to 40 elements, and ◇ to 1200 elements on the mesh.*

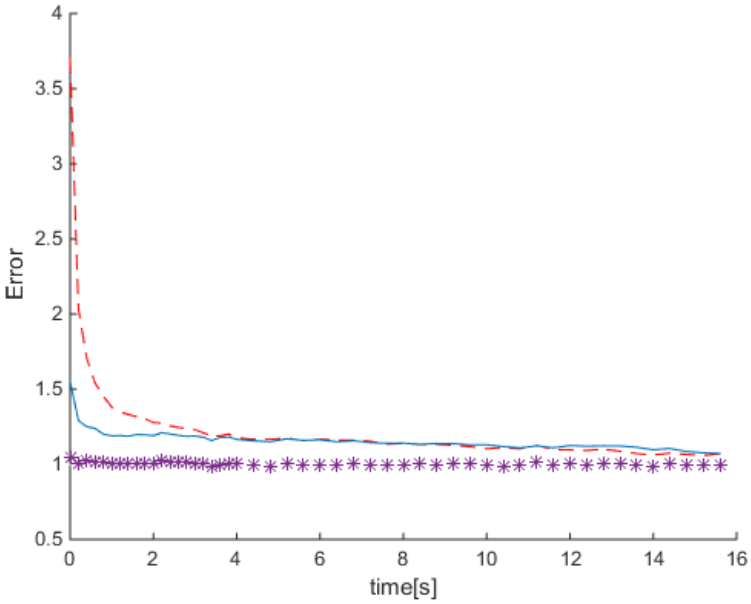

*Figure 35: Comparison of recovery curves divided by recovery curve of 1200 mesh elements near the boundary approximated by different number of elements. Solid line corresponds to a mesh of 4 elements, dashed line to 12 elements, and * to 40 element on the mesh.*

Although the mesh detail should be geometry specific, this analysis demonstrates that at least 40 mesh triangles are necessary to avoid distortion shown in Figures 31 and 32.

3.3.3 Time step Duration

We ran simulations with varying time step sizes 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1} and the results are shown in Figure 35.

The settings used in the simulations can be found in Appendix 4.

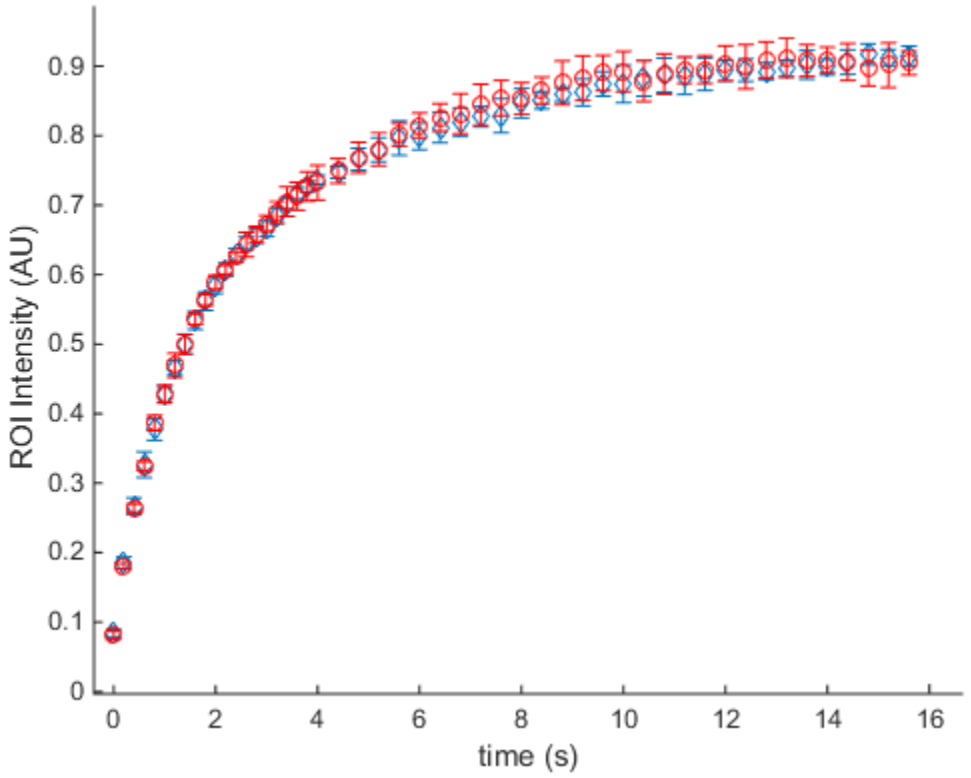

***Figure 36**. Comparison of recovery curves generated by different time step sizes. o is 0.1s, ◇ is 0.0001s.*

We conclude that time step lengths up to $1*10^{-1}$s are stable.

4 APPLICATION TO TIP GROWING PLANTS

The simulation package we developed was used to examine the boundary effects on recovery curves, particularly for different plant cell geometries. Many plants have different shapes of their tip [10] and to mimic these geometries we used five "idealistic realistic" geometries. These were a cylinder, a cylinder with spherical cap attached to it (Bullet, Regular), a cylinder with narrow elliptical cap (Bullet, Short), a cylinder with wide elliptical cap (Bullet, Long) and a cylinder with conic cap attached to it. All of the geometries have a total length of 25 μm and radius of 5 μm (see Figure 37). The cell shape was modeling as an ellipsoid, i.e.

$$\left(\frac{x}{a}\right)^2+\left(\frac{y}{b}\right)^2+\left(\frac{z}{c}\right)^2=1 \qquad (11)$$

and the sharpness was controlled by changing the exponents to higher order. Once the idealized shape was made, it was converted to a mesh. For each of the geometries five ROIs were simulated 3 small (r=1 μm) with varying x locations: 12.5 μm, 20 μm and 24 μm; and 2 large (r=5 u μm) simulated at varying x locations: 12.5 μm and 20 μm (see Figure 38).

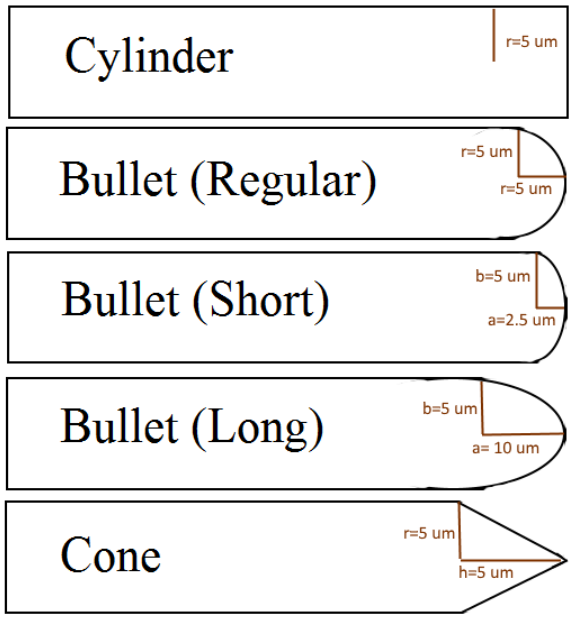

Figure 37. *Idealistic geometries used.*

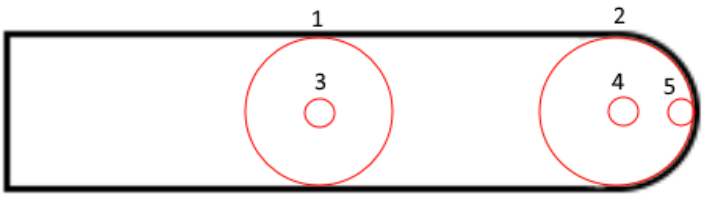

Figure 38*. ROI locations.*

Simulations were run with parameters specified in Appendix 4, and recovery curves were generated. Despite not being appropriate to describe such a geometry, recovery curves are often fitted to either the open boundary solution (Eq. (3)) or a double exponential function. We followed a similar approach here to determine the range of validity of these approaches.

The recovery curves generated were first fitted to the open boundary solution where bleach fraction, maximum value (constant multiple of Eq. (3)) and diffusion coefficient were chosen as free parameters. The diffusion coefficients were then plotted grouped by ROI (see Figure 39). Note that the input diffusion coefficient was 1 $\mu m^2/s$ in all the simulations.

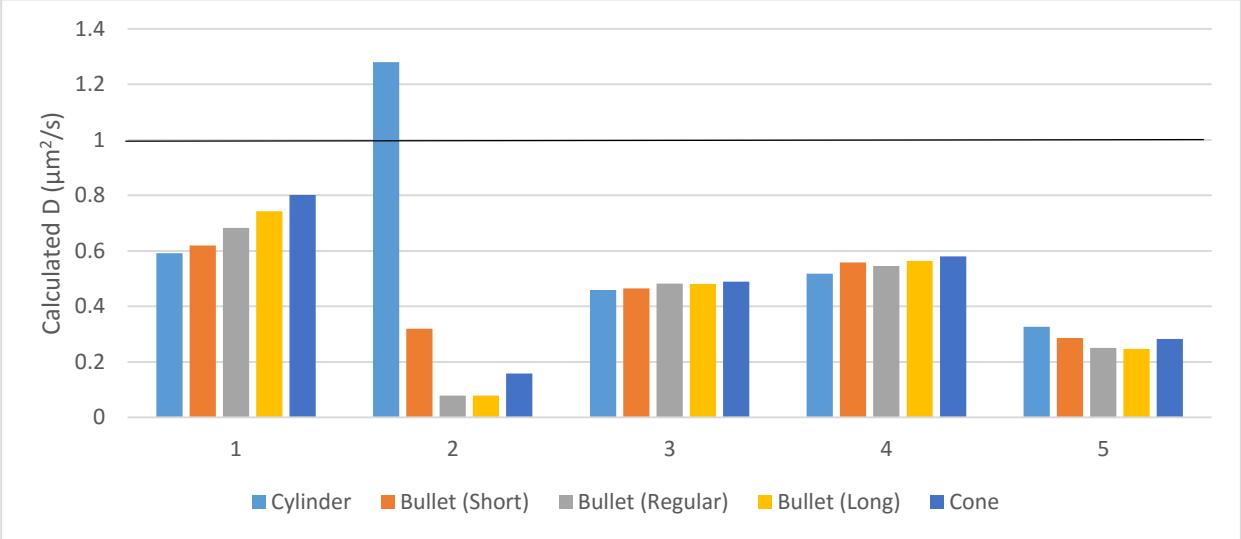

***Figure 39**. D values estimated using the open boundary fit. Bars are in order given by the legend, numbers correspond to different ROI locations shown in Figure 38.*

As seen in Figure 39, the greatest variance is for the large ROIs starting with the one at the edge. For the small ones the variance increases as ROI gets closer to the tip. This is most likely due to portion of the bleach beam that is outside the geometry.

The recovery curves can also be fitted to double exponential function, as commonly done in the literature, i.e.

$$f(t) = C_1 + C_2 e^{-t/\tau_1} + C_3 e^{-t/\tau_2}, \qquad (32)$$

where C_1, C_2 and C_3 are coefficients and τ_1 and τ_2 are characteristic times. Characteristic time can be converted to diffusion coefficient via

$$D = \frac{r^2}{\tau}. \qquad (33)$$

We performed the fits to a double exponential function and obtained the slow and fast diffusion coefficients (corresponding to fast and slow characteristic times), and the results are shown in Figures 40 and 41.

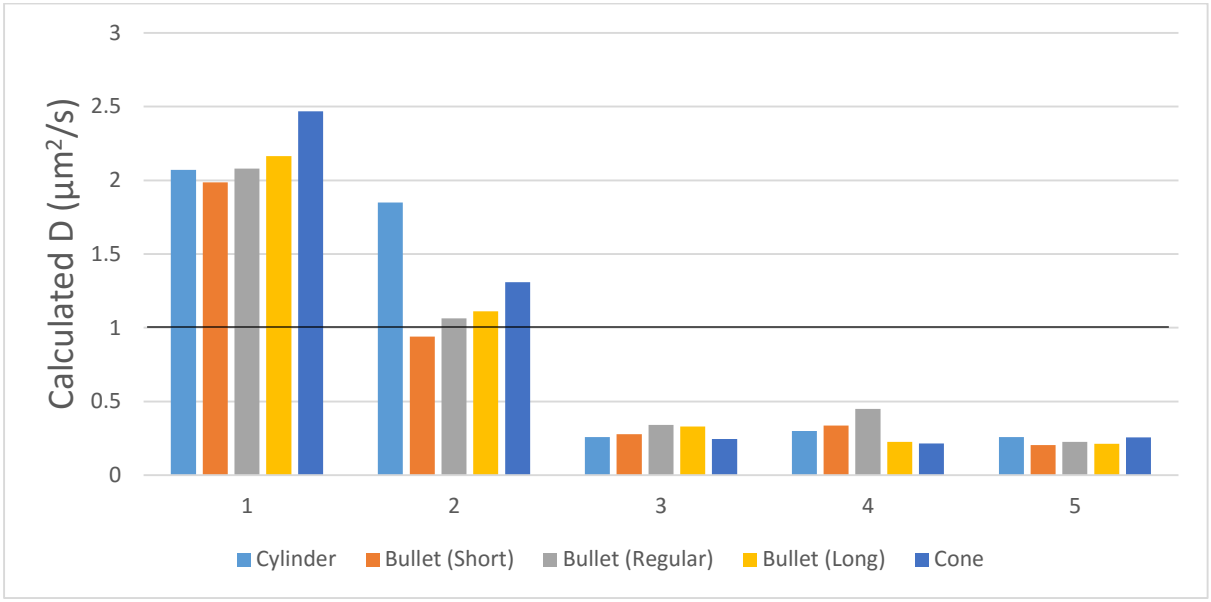

Figure 40. *D values estimated from the slow characteristic time sorted by different ROI sizes. Bars are in order given by the legend.*

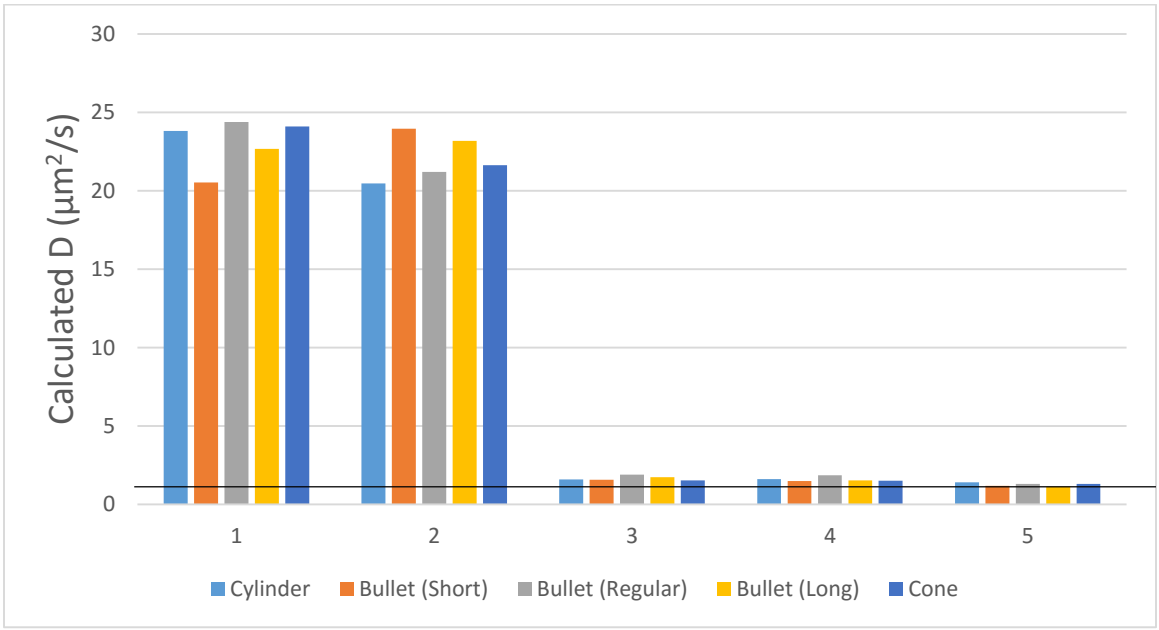

Figure 41. *D values estimated from the fast characteristic time. Bars are in order given by the legend.*

As seen in Figures 40 and 41 it is extremely difficult to consistently get a diffusion coefficient from this fit. Note that, only one diffusion coefficient is input to the simulation (D = 1 $\mu m^2/s$). The double exponential fit gives two diffusion coefficients. This suggests two mechanistic process. This is physically implausible because there is only one type of molecule diffusing.

5 SUMMARY AND FUTURE DIRECTIONS

FRAP is a technique that uses photobleaching to observe diffusion inside cells. The program in this project uses Brownian motion for each particle in a data parallel way on GPU to simulate the motion. It uses Gaussian beam to simulate the confocal microscope configuration. Results from different ROIs and geometries were analyzed. It was determined that geometry effects the recovery curve more as ROI gets bigger and closer to boundary. Commonly used analytical models and double exponential fits to FRAP recovery curves do not consistently yield correct diffusion coefficients and are strongly discouraged to be used.

For future work, we suggest to include the photobleaching effect caused by the imaging process, include non-circular ROI shapes, extend to other microscope systems, estimate diffusion coefficient for a given set of experimental images using error minimization and compare FRAP to other techniques such as Fluorescence Correlation Spectroscopy (FCS) and Single-Particle tracking.

6 REFERENCES

[1] J. B. Nikos Alexandratos, "WORLD AGRICULTURE TOWARDS 2030/2050," Food and Agriculture Organization of the United Nations, 2012.

[2] R. Phillips, J. Kondev and J. T. H. G. Garcia, "A Statistical View of Biological Dynamics," in *Physical Biology of the Cell*, London and New York, Garland Science, 2013, pp. 509-542.

[3] J. Bibeau, J. Kingsley, Z. Chen, X. Huang, E. Tüzel and L. Vidali, "Experimental and computational FRAP reveals the underestimated role of vesicle transport via diffusion in polarized growth," *In Preparation.*

[4] J. Howard, "Dynamics of Bending and Buckling," in *Mechanics of Motor Proteins and the Cytoskeleton*, Sunderland, MA, Sinauer Associates, 2001, pp. 106-109.

[5] H. C. Ottinger, "Stochastic Calculus," in *Stochastic Processes in Polymeric Fluids Tools and Examples for Developing Simulation Algorithms*, Berlin Heidelberg, Springer-Verlag, 1996, pp. 81-146.

[6] Leica Microsystems, *Leica TCS SP5 User Manual*, Mannheim.

[7] M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, Washington DC: National Bureu of Standards, 1972.

[8] aparapi, "What is Aparapi?," [Online]. Available: https://code.google.com/archive/p/aparapi/. [Accessed 20 April 2016].

[9] M. Burns, "The StL Format," in *Automated Fabrication: Improving Productivity in Manufacturing*, 1993.

[10] O. Campas, E. Rojas, J. Dumais and L. Mahadevan, "Strategies for Cell ShapeControl in Tip-Growing Cells," *American Journal of Botany*, vol. 99, no. 9, pp. 1577-1582, 2012.

Appendix 1a: Recipe

Tif Stack to STL Mesh

1. Drag and drop the TIF stack for the cell that was generated by the microscope to imagej.
2. Use Process->Filters->Gaussian Blur->2.00 to blur the image.
3. Use Image->Adjust-> Threshold to pick the correct range of intensities that will be included in the mesh.
4. Plugins->3D->3D Viewer
5. Display as->Surface
6. File->Export Surfaces -> STL (binary)

Mesh simplification

1. Open the STL file with MeshLab.
2. Filters->Normals->Curvatures and Orientation->Re-orient faces coherently
3. If the faces are black: Filters->Normals->Curvatures and Orientation->Invert Faces Orientation
4. File->Export Mesh->Binary Encoding->OK

Settings input

A template of configuration file can be found in Appendix 1b. Fill in values of the fields in. Save it as "FrapSettings.txt" in the directory of the executable. The first portion of this file are settings. "meshFile" and "folderName" are strings all others are numbers. The line order does not matter as long as all the settings are above the "-END SETTINGS-" line. For a value to be taken as a command line input type "commandLine" for the field and in the command line enter –[fieldName] followed by the value.

Type the event sequence between "-END SETTINGS-" and "-END-". There are 3 types of events: movement, imaging and bleaching represented by 'm', 'i' and 'b' respectively. The syntax for event input is: "[type] [repetition] [seconds of delay in between]". Use one event per line.

Appendix 1b: Example Settings File

```
1 = diffusionCoefficient
commandLine = meshFile
1e-5 = scaleFactor
0.850 = bleachW0
250 = xPixels
100 = yPixels
commandLine = folderName
0.875 = bleachZr
0.01 = bleachProbability
commandLine = numberParticles
0.350 = imageW0
0.875 = imageZr
1e-2 = maxTimeStepLength
```

3.57e-4 = imageDelay
3.57e-4 = bleachDelay
243 = bleachLines
commandLine = bleachCenterx
0.5 = bleachCentery
0.5 = bleachCenterz
commandLine = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 1
b 2 0.125
i 27 0.113
i 1 0.199
i 30 0.250
i 9 1
-END-

Appendix 2: Tables of Calculated Diffusion Coefficients

ROI / Mesh	1	2	3	4	5
Cylinder	0.59	1.28	0.46	0.52	0.33
Bullet (Short)	0.62	0.32	0.46	0.56	0.29
Bullet (Regular)	0.68	0.08	0.48	0.55	0.25
Bullet (Long)	0.74	0.08	0.48	0.56	0.25
Cone	0.80	0.16	0.49	0.58	0.28

Table 1: D from open boundary fit $\mu m^2/s$.

ROI / Mesh	1	2	3	4	5
Cylinder	2.07	1.85	0.26	0.30	0.26
Bullet (Short)	1.99	0.94	0.28	0.34	0.21
Bullet (Regular)	2.08	1.06	0.34	0.45	0.23
Bullet (Long)	2.16	1.11	0.33	0.23	0.21
Cone	2.47	1.31	0.25	0.22	0.26

Table 2: D from large characteristic time $\mu m^2/s$.

ROI / Mesh	1	2	3	4	5
Cylinder	23.81	20.45	1.59	1.61	1.40
Bullet (Short)	20.53	23.96	1.56	1.48	1.18
Bullet (Regular)	24.39	21.21	1.90	1.85	1.30
Bullet (Long)	22.66	23.18	1.74	1.54	1.15
Cone	24.11	21.62	1.53	1.50	1.30

Table 3: D from small characteristic time $\mu m^2/s$.

Appendix 3: STL Input Endian Conversion

```
next = 0;
next |= input.read();
next |= (input.read() << 8);
next |= (input.read() << 16);
next |= (input.read() << 24);
bn[i] = Float.intBitsToFloat(next) * scale;
```

Appendix 4: Input Parameters

Section 3.1.1

1 = diffusionCoefficient
Meshes/pancake.stl = meshFile
1e-6 = scaleFactor
0.0223 = bleachW0
1000 = xPixels
1000 = yPixels
commandLine = folderName
1000 = bleachZr
1 = bleachProbability
300000 = numberParticles
.0223 = imageW0
1000 = imageZr
1e-2 = maxTimeStepLength
4e-10 = imageDelay
4e-10 = bleachDelay
200 = bleachLines
0.5 = bleachCenterx
0.5 = bleachCentery
0.5 = bleachCenterz
2 = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 0.1
b 2 0.01
i 20 0.2
i 30 0.4
-END-

Section 3.1.2

1 = diffusionCoefficient
Meshes/bullet.stl = meshFile
1e-6 = scaleFactor
0.850 = bleachW0
250 = xPixels
100 = yPixels
commandLine = folderName
0.875 = bleachZr

0.01 = bleachProbability
1000000 = numberParticles
0.350 = imageW0
0.875 = imageZr
1e-2 = maxTimeStepLength
3.57e-4 = imageDelay
3.57e-4 = bleachDelay
243 = bleachLines
0.92 = bleachCenterx
0.5 = bleachCentery
0.5 = bleachCenterz
2 = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 1 1
b 2 0.125
i 27 0.113
i 1 0.199
i 30 0.250
i 9 1
-END-

Section 3.3.1

1 = diffusionCoefficient
Meshes/pancake.stl = meshFile
1e-6 = scale
0.223 = bleachW0
200 = xPixels
200 = yPixels
commandLine = folderName
1 = bleachZr
1 = bleachProbability
commandLine = size
.5 = imageW0
.7 = imageZr
1e-2 = maxTimeStepLength
4e-4 = imageDelay
4e-4 = bleachDelay
0.5 0.5 0.5 = belachCenterXYZ
2 = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 0.1
b 2 0.1
i 20 0.2
i 30 0.4
-END-

Section 3.3.2

1 = diffusionCoefficient
commandLine = meshFile
1e-6 = scale
0.223 = bleachW0
200 = xPixels
200 = yPixels
commandLine = folderName
1 = bleachZr
1 = bleachProbability
100000 = size
.5 = imageW0
.7 = imageZr
1e-2 = maxTimeStepLength
4e-4 = imageDelay
4e-4 = bleachDelay
0.5 0.5 0.5 = belachCenterXYZ
2 = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 0.1
b 2 0.1
i 20 0.2
i 30 0.4
-END-

Section 3.3.3

1 = diffusionCoefficient
Meshes/pancake.stl = meshFile
1e-6 = scale
0.223 = bleachW0
200 = xPixels
200 = yPixels
commandLine = folderName
1 = bleachZr
1 = bleachProbability
100000 = size
.5 = imageW0
.7 = imageZr
commandLine = maxTimeStepLength
4e-4 = imageDelay
4e-4 = bleachDelay
0.5 0.5 0.5 = belachCenterXYZ
2 = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 0.1
b 2 0.1
i 20 0.2

i 30 0.4
-END-

Section 4

1 = diffusionCoefficient
commandLine = meshFile
1e-5 = scaleFactor
0.850 = bleachW0
250 = xPixels
100 = yPixels
commandLine = folderName
0.875 = bleachZr
0.01 = bleachProbability
commandLine = numberParticles
0.350 = imageW0
0.875 = imageZr
1e-2 = maxTimeStepLength
3.57e-4 = imageDelay
3.57e-4 = bleachDelay
243 = bleachLines
commandLine = bleachCenterx
0.5 = bleachCentery
0.5 = bleachCenterz
commandLine = bleachRadius
0.5 = imageZ
-END SETTINGS-
i 10 1
b 2 0.125
i 27 0.113
i 1 0.199
i 30 0.250
i 9 1
-END-